

找不到工作怎么办？毕业就失业怎么办？人才市场中人山人海，我怎样才能脱颖而出，成为企业青睐的新人？

好不容易找到工作了，我怎样才能成为闪耀的职场新星？我怎样才能获得同事的支持，领导的重用？我怎样才能拥有卓越的能力，成就自己的理想？

本书中，作者总结自身十多年的软件开发行业工作经验，对这些问题做出探讨，对毕业生和职场菜鸟们提出一些实用的建议，帮助菜鸟们在职场上顺利腾飞。



袁永福
编著



从毕业生 到程序员

使用C#开发商业软件



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>



关于作者

袁永福，2001年从南京东南大学毕业，Visual C#技术方面的微软 MVP，有着十多年的软件开发工作经验。一直在图形软件开发、XML/XSLT、软件架构设计等领域进行着深入的探索和实践，并主导开发出通用报表引擎、文本编辑器等产品，对复杂图形类软件的架构设计、开发和产品化工作有着很高的水平，现从事着电子病历相关开发。作者博客：<http://www.cnblogs.com/xdesigner>。



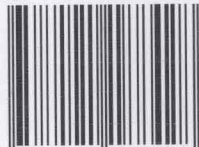
@博文视点Broadview



责任编辑：孙学瑛
封面设计：张跃

上架建议：软件工程

ISBN 978-7-121-18842-8



9 787121 188428 >

定价：69.00元

更多资源请访问稀酷客(www.ckook.com)

从毕业生到程序员

使用C#开发商业软件



袁永福
编著



电子工业出版社

Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书的初衷就是帮助企业建立新人在入职 2 年多的时间中关于解决职场新人面临的编程技术技能缺乏、软件行业思想、职场规划等很多问题的系统培养理念和制度。因学生在学校里开发的软件是学习和学术性的软件，本书也能帮助毕业生们能尽快地从学习型的软件开发转换为商业性的软件开发，尽早形成实际生产力，这对毕业生和企业都是有好处的。

本书并不想成为面试宝典之类的书；面试宝典是应试教育在企业面试中的延伸，只是为了应付几十分钟的面试考试，治标不治本。本书确实想帮助毕业生顺利地度过两年的程序员职场新手阶段，让更多的职场新星确定正确的方向，少走弯路，早日腾飞。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

从毕业生到程序员：使用 C# 开发商业软件 / 袁永福编著. —北京：电子工业出版社，2013.1
ISBN 978-7-121-18842-8

I. ①从… II. ①袁… III. ①C 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字（2012）第 257639 号

责任编辑：孙学瑛

特约编辑：伦朝丽

印 刷：北京东光印刷厂

装 订：三河市皇庄路通装订厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：29.25 字数：723 千字

印 次：2013 年 1 月第 1 次印刷

印 数：3500 册 定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zltts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

PREFACE

前言

随着高等院校的扩招，高校毕业生也越来越多，此时大学生从过去的天之骄子跌落成为如今的芸芸众生，这是一个残酷的现实。社会新增的工作机会不多，而毕业生的就业需求日趋增大，此时应届毕业生、往届毕业生、已有工作经验人士等，数百万的人涌入人才市场追逐着相对短缺的就业机会，竞争日益激烈。

更进一步的，很多企业不想直接招聘高校毕业生，在当前巨大的成本压力下，宁愿高价招聘具有工作经验的做事可靠的求职者，也不愿意招聘价格相对低廉的高校毕业生。于是形成了一边是大量的高校毕业生找不到合意的工作，甚至长期处于失业状态；而另外一边企业招不到中意的人才。这是一个不小的社会矛盾，既抬高了失业率，又造成了教育资源的大量浪费。

高校毕业生的就业问题由来已久，随着近年来全球范围的金融危机乃至实体经济危机的影响，中国社会出现一些变化和转型，此时这个问题凸显出来，成为一个广受关注的社会问题。

从经济的角度上看，高校毕业生就业问题是一种商品供应错位的问题。

高校和企业存在供需关系。高校培养毕业生，满足企业的用人需求。而企业接受这些毕业生，满足其自身的用人需求。因此高校的核心工作应该是满足企业的用人需求，高校的核心利益是建立在企业用人需求的基础上。受到企业欢迎的高校是成功的高校，必然能得到长久持续的良好发展；而没得到企业认同的高校是不成功的高校，也不会有长久的经济效益。

然而，现实情况刚好相反，一些高校并不是以企业的用人需求为核心工作目标，而是热衷于教育产业化，无论是学校管理层还是教师个人，都在寻求短期的经济利益，并且以各种手段快速地提高表面就业率。这最终不会为企业所认同，造成供需错位，损害了双方的利益，加剧了高校毕业生就业问题。

要比较好地解决这个问题，得从根本上解决当代高校教育机制以及社会人力资源利用机制，不过这方面超出笔者的能力范围，因此不想细说。

笔者只想在能力范围内尽量做出一些工作，那就是回顾个人 11 年的计算机软件开发行业工作经验，发现问题，分析问题，多方比较，为缓解毕业生就业问题提出一些实用可行的建议。希望能帮助毕业生提高人才市场竞争力，顺利地进入职场并可持续性的发展。

笔者毕业后一直从事着计算机软件开发工作，使用了 C#编程语言。因此根据笔者的个人能力，对有志于进入计算机软件开发的高校毕业生和新手提出一些经验的总结，并以 C#语言为核心展开了一些技术培训内容，包括软件开发和管理的思想，C#的运用，并介绍了一些在当前和未来一段时间内都很实用的技术，使得菜鸟们能较为迅速地胜任 C#软件工程师的角色，从而开始了使用 C#开发商业软件的职场之路。

笔者帮助菜鸟的同时，希望也能对软件开发企业有好处，希望此书能帮助企业对接收过来的毕业生和菜鸟们顺利地展开训练，使得他们能尽快拥有实际生产力，迅速进入工作状态，从而使这些新人早日给企业创造经济效益，实现企业的利益最大化。

本书并不想成为面试宝典之类的速成书籍，因为笔者认为，卓越的能力是长期扎实的工作锻炼出来的，高超的技艺是通过经常性的学习和独立思考而造就的，突出的贡献是紧密的团队共同协作出来的。

在此特别感谢一些人，包括柯凌云，她根据自己从事高校教育的宝贵工作经验，给我带来了启发；陈朕，他对 C#以及数据库开发的部分提供了一些信息；王清培，新科微软 MVP，也对这部书提出了一些意见。有了这些人的帮助，使得这本书增色不少，在此深表感谢。

在本书的编写过程中，参考了许多相关的书籍、资料和互联网发布的信息，编者在此对这些参考资料的作者表示感谢。同时还要感谢电子工业出版社在本书出版过程中给予的支持和帮助。

高校毕业生就业和培训问题是一个复杂的问题，因笔者水平能力有限，书中难免存在错漏和不妥之处，望读者指正，以利改进和提高，帮助作者进步。

袁永福
2012 年 11 月于南京

博文视点诚邀精锐作者加盟

《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、
《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

九年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金的
计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者
朋友加盟,与大师并列于IT专业出版之巅峰。

英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

· 专业的作者服务 ·

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们

博文视点官网: <http://www.broadview.com.cn>

新浪官方微博: <http://weibo.com/broadviewbj>

投稿电话: 010-51260888 88254368

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

腾讯官方微博: <http://t.qq.com/bowenshidian>

投稿邮箱: jsj@phei.com.cn

关于本书用纸的温馨提示

亲爱的读者朋友: 您所拿到的这本书使用的是 **环保轻型纸**!

环保轻型纸在制造过程中添加化学漂白剂较少,颜色更接近于自然状态,具有纸质轻柔、光反射率低、保护读者视力等优点,其成本略高于胶版纸。给您带来更好的阅读体验并与读者共同支持环保,我们在没有提高图书定价的前提下,使用这种纸张。愿我们共同分享纸质图书的阅读乐趣!

CONTENTS

目 录

第 1 章 一毕业就面临的问题.....	1
1.1 高校毕业生就业途径.....	1
1.2 计算机专业毕业生的就业问题.....	3
1.3 求职之路.....	4
1.3.1 抉择.....	4
1.3.2 兴趣和志向.....	4
1.3.3 顺趋势者昌，逆趋势者亡.....	5
1.4 中国软件行业的特点和趋势.....	6
1.4.1 盗版.....	7
1.4.2 嵌入式开发.....	7
1.4.3 移动开发.....	8
1.4.4 SAAS 及云计算.....	9
1.5 行业应用软件开发.....	11
1.6 求职战略方向.....	12
1.7 求职战术手段.....	12
1.7.1 接触用人企业.....	12
1.7.2 求职过程.....	13
第 2 章 初入职场.....	15
2.1 从学生期到职场期的变轨.....	15
2.1.1 人生的节日.....	16
2.1.2 团队意识.....	16
2.2 积累与创新.....	22
2.3 技术之害.....	24

- 2.4 关于薪酬 25
- 2.5 关于买房 26
- 2.6 关于自信心 28
- 2.7 小结 28
- 第 3 章 商业软件开发基础 29
 - 3.1 学习型软件开发和商业软件开发 29
 - 3.2 项目软件和产品软件 29
 - 3.2.1 合同软件 30
 - 3.2.2 产品软件 30
 - 3.2.3 自营软件 31
 - 3.3 商业软件开发基本概念 31
 - 3.4 商业软件开发的平衡点 31
 - 3.4.1 软件开发时间 32
 - 3.4.2 软件质量 32
 - 3.4.3 软件功能 33
 - 3.5 过程和质量控制 33
 - 3.6 新旧技术的权衡 34
 - 3.7 商业包装 34
 - 3.8 商业软件开发人员的工作环境 36
 - 3.8.1 对于客户 36
 - 3.8.2 对于市场销售人员 37
 - 3.8.3 对于管理层 37
- 第 4 章 开发者眼里的 Windows 39
 - 4.1 Windows Service 39
 - 4.2 管理 Windows Service 40
 - 4.3 事件查看器 42
 - 4.4 远程桌面 44
 - 4.4.1 配置服务器 44
 - 4.4.2 连接远程桌面 45
 - 4.5 任务管理器 47
 - 4.5.1 管理应用程序 48
 - 4.5.2 管理进程 49
 - 4.5.3 查看系统性能 50
 - 4.5.4 查看网络 50
 - 4.5.5 管理正在登录的用户 51

第 5 章 C#程序开发 52

5.1 C#语言简介 52

5.2 .NET 框架简介 53

5.2.1 托管应用程序 54

5.2.2 微软中间语言规范 55

第 6 章 C#基本语法 57

6.1 C#应用系统模块逻辑框架 57

6.1.1 应用系统 57

6.1.2 程序集 57

6.1.3 命名空间 58

6.1.4 类型 59

6.1.5 类型成员 60

6.1.6 功能语法块 60

6.2 数据类型 61

6.2.1 基础数据类型 61

6.2.2 数组 63

6.2.3 自定义类型 64

6.3 数据类型转换 72

6.3.1 强制类型转换 72

6.3.2 as 类型转换 73

6.3.3 is 类型判断 74

6.4 可访问级别 74

6.4.1 private 私有的 75

6.4.2 protected 受保护的 75

6.4.3 internal 内部的 75

6.4.4 public 公开的 76

6.5 类型样式 76

6.5.1 static class 静态类 76

6.5.2 abstract class 抽象类 77

6.5.3 sealed class 密封类 78

6.6 类型成员 79

6.6.1 构造函数 80

6.6.2 字段 80

6.6.3 属性 81

6.6.4 方法 81

6.6.5 事件 82

6.6.6	索引器	82
6.6.7	静态成员	84
6.6.8	实例成员	86
6.6.9	常数成员	87
6.7	面向对象编程	87
6.7.1	类	87
6.7.2	封装	87
6.7.3	继承	88
6.7.4	重载	88
6.7.5	重写	89
6.8	表达式	90
6.8.1	数学表达式	90
6.8.2	逻辑表达式	91
6.8.3	位运算表达式	91
6.9	执行结构	92
6.9.1	顺序执行	92
6.9.2	条件判断	92
6.9.3	循环结构	95
6.9.4	异常处理结构	99
6.10	C#转型建议	103
6.10.1	从 VB 到 C#	103
6.10.2	从 C/C++ 到 C#	105
6.10.3	从 Java 到 C#	107
第 7 章	第一次 C# 体验	108
7.1	第一次使用 VS.NET 集成开发环境	108
7.1.1	菜单栏	109
7.1.2	工具条	109
7.1.3	工具箱	109
7.1.4	主工作区	109
7.1.5	解决方案资源管理器	109
7.1.6	属性编辑器	110
7.2	C# 程序类型	112
7.2.1	Windows 应用程序	113
7.2.2	ASP.NET Web 应用程序	113
7.2.3	命令行应用程序	113
7.2.4	其他类型的应用程序	114

第 8 章 开发第一个 Windows 应用程序 115

8.1 建立 C# Windows 应用程序项目 115

8.2 WinForm 控件工具箱 116

8.3 WinForm 窗体设计器 118

8.4 Windows 窗体设计概念及原则 122

8.5 Main 函数 130

8.6 解决方案资源管理器 132

8.7 解决方案资源树状列表 132

8.7.1 解决方案 132

8.7.2 程序工程 133

8.7.3 引用 135

8.7.4 程序文件 136

8.7.5 文件夹 137

8.8 解决方案资源管理工具条 139

8.8.1 属性按钮 139

8.8.2 添加新解决方案文件夹按钮 142

8.8.3 显示所有文件按钮 142

8.8.4 刷新按钮 143

8.9 控件属性编辑区域 144

8.9.1 控件名称下拉列表 144

8.9.2 属性列表工具条 145

8.9.3 属性项目列表 147

8.10 设计用户界面 150

8.11 用户界面事件处理原理 154

8.11.1 鼠标事件 155

8.11.2 键盘事件 156

8.12 编写事件处理代码 156

8.12.1 读写系统配置 159

8.13 调试 161

8.13.1 执行代码 161

8.13.2 查看和修改变量值 163

8.13.3 命令窗口 165

8.14 测试和运行 Windows 应用程序 165

8.15 小结 166

第 9 章 开发第一个 ASP.NET 应用程序 167

9.1 ASP.NET 概念 167

9.1.1	B/S 架构	167
9.1.2	HTTP 传输协议	169
9.1.3	HTML 文档规范	173
9.1.4	ASP.NET 服务器端架构	174
9.1.5	ASP.NET Web 服务器控件技术	178
9.1.6	ASP.NET 客户端开发架构	182
9.2	建立 C# ASP.NET 应用程序项目	183
9.3	ASP.NET 页面设计器	186
9.3.1	ASP.NET 页面工具箱	186
9.3.2	Web 页面内容编辑器	187
9.4	ASP.NET 控件工具箱	188
9.5	设计用户界面	189
9.6	编写后台代码	189
9.6.1	输出 JavaScript	192
9.7	测试和运行 ASP.NET 应用程序	196
9.8	部署 ASP.NET 应用程序	196
9.8.1	准备运行环境	197
9.8.2	准备应用程序目录	197
9.8.3	创建虚拟目录	197
9.8.4	配置虚拟目录	201
9.9	其他部署相关技术手段	204
9.9.1	Aspnet_regiis.exe	204
9.9.2	IISReset.exe	204
9.9.3	配置 ASP.NET 账号权限	204

第 10 章 开发第一个 Web Service 程序.....207

10.1	Web Service 原理	207
10.2	软件功能需求	208
10.3	建立 C# Web Service 应用程序项目	208
10.4	编写 Web 方法	210
10.5	发布 Web Service	216
10.6	使用 Web Service	216
10.7	在 Windows 应用程序中使用 Web Service	216
10.7.1	添加 Web 引用	216
10.7.2	使用 Web Service	217
10.8	在 ASP.NET 应用程序中使用 Web Service	222
10.9	Web Service 原理	225

第 11 章 开发第一个 ADO.NET 数据库应用程序..... 228

11.1 ADO.NET 数据库访问概述 228

11.2 建立 C#应用程序项目 229

11.2.1 快速读取数据 229

11.2.2 数据源绑定 233

11.2.3 修改数据 236

11.3 类型使用参考说明..... 253

11.3.1 System.Data.IDbConnection 接口类型 253

11.3.2 System.Data.IDbCommand 接口类型 253

11.3.3 System.Data.IDataReader 接口类型 254

11.3.4 System.Data.IDataParameter 接口类型..... 255

11.3.5 System.Data.DataTable 类型 256

11.3.6 System.Data.DataColumn 类型 257

11.3.7 System.Data.DataRow 类型 258

第 12 章 开发第一个 JavaScript 应用程序 259

12.1 JavaScript 基本概念 259

12.2 HTML DOM 261

12.3 JavaScript 语法 263

12.3.1 数据类型 263

12.3.2 运算符 264

12.3.3 条件判断语法结构 266

12.3.4 循环语法结构 267

12.3.5 异常处理语法结构 268

12.3.6 其他语法结构 270

12.4 系统预定义对象 271

12.4.1 系统实例对象 272

12.4.2 系统全局对象 274

12.5 JavaScript 代码文件 282

12.5.1 文本编码格式 283

12.6 JavaScript 调试 285

12.6.1 设置 IE 285

12.6.2 插入断点 285

12.6.3 调试 287

12.7 JavaScript 应用实例 288

12.7.1 走马灯 288

12.7.2 网页对话框 289

12.7.3	日历对话框	293
12.7.4	数据验证	294
12.8	浏览器兼容性	301
12.9	小结	302
第 13 章	开发第一个 XML 应用程序	303
13.1	XML 应用框架	303
13.2	XML 的发展历史	304
13.3	XML 基础知识介绍	305
13.3.1	XML 基本语法知识	305
13.3.2	W3C 国际标准组织	307
13.3.3	国际标准的意义	307
13.4	微软.NET 框架对 XML 的支持	308
13.4.1	流式处理模型	308
13.4.2	DOM 处理模型	308
13.5	输出 XML 文档	310
13.5.1	PageUseXmlTextWriter.aspx	311
13.5.2	PageUseXmlDocument.aspx	316
13.6	类型使用参考说明	318
13.6.1	System.Xml.XmlTextWriter 类型	318
13.6.2	System.Xml.XmlDocument 类型	319
13.6.3	System.Xml.XmlElement 类型	320
13.7	小结	321
第 14 章	开发第一个文件系统操作应用程序	322
14.1	文件系统操作概述	322
14.1.1	文件和目录	322
14.1.2	文件路径	322
14.1.3	文本文件和二进制文件	323
14.1.4	.NET 的文件系统开发	323
14.2	建立 C#应用程序项目	323
14.2.1	设计主窗体	323
14.2.2	浏览目录	325
14.2.3	浏览文件	329
14.2.4	查看、编辑文本内容	331
14.2.5	查看图片内容	339
14.2.6	访问文件内容	342

第 15 章 关系型数据库开发基础.....344

15.1 主流数据库介绍.....344

15.1.1 MS Access.....344

15.1.2 MS SQL Server.....353

15.2 SQL 语言.....372

15.2.1 查询数据.....373

15.2.2 新增数据.....377

15.2.3 修改数据.....378

15.2.4 删除数据.....378

15.2.5 视图.....379

15.2.6 存储过程.....380

15.2.7 触发器.....381

第 16 章 商业软件开发规范.....382

16.1 C#代码书写规范.....382

16.1.1 代码缩进.....382

16.1.2 空行.....383

16.1.3 换行.....383

16.1.4 空格.....385

16.1.5 定义类型.....385

16.1.6 大小写.....387

16.1.7 名称.....388

16.1.8 名称空间.....390

16.1.9 语句.....391

16.1.10 注释.....391

16.1.11 代码文件目录结构.....392

16.2 C#软件开发原则.....392

16.2.1 尽晚创建，尽早释放.....392

16.2.2 单入口，单出口.....394

16.2.3 最小权限原则.....395

16.2.4 尽早暴露错误原则.....397

第 17 章 面向对象软件开发方法.....401

17.1 发现问题.....401

17.2 分析问题.....401

17.3 解决问题.....402

17.3.1 任务分解.....402

- 17.3.2 过程控制403
 - 17.3.3 知识重用403
 - 17.3.4 代码重用405
- 17.4 面向对象开发406
 - 17.4.1 封装406
 - 17.4.2 继承407
- 第 18 章 团队开发管理408**
 - 18.1 项目管理408
 - 18.1.1 项目启动409
 - 18.1.2 需求开发409
 - 18.1.3 项目计划410
 - 18.1.4 系统设计410
 - 18.1.5 开发实施412
 - 18.1.6 系统测试412
 - 18.1.7 发布部署413
 - 18.1.8 试用验收413
 - 18.1.9 项目结项413
 - 18.1.10 项目移交414
 - 18.1.11 项目管理415
 - 18.1.12 QA415
 - 18.1.13 CM415
 - 18.2 源代码管理416
 - 18.2.1 源代码管理的原理416
 - 18.2.2 VSS 源代码管理软件419
 - 18.2.3 使用 VSS 客户端软件419
 - 18.2.4 在 VS.NET 中使用 VSS435
- 附录 A 关于企业培训442**
- 附录 B 关于盗版445**

一毕业就面临的问题

高校毕业生就业问题已经存在了一段时间，据中国人力资源和社会保障部官方数据“我国大学毕业生从 2000 年的 107 万人到 2009 年跃升为 610 万人，增长了近 6 倍，加上历年没有就业人员，2009 年需要安置的学生为 700 多万。2010 年全国高校毕业生为 630 万人。”

图 1-1 列出了中国近几年的高校毕业生人数。

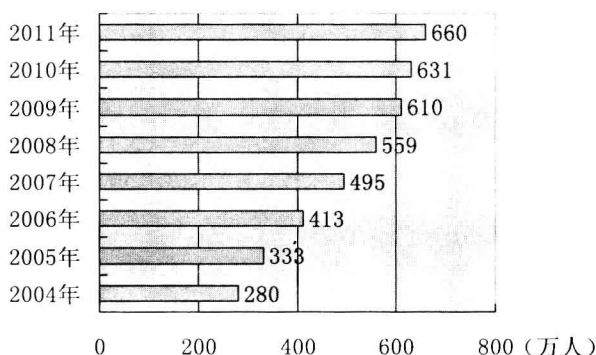


图 1-1 2004—2011 年中国高校毕业生人数统计表

这些官方数据说明现在大学生的就业问题是非常严重的，更进一步地，2010 年 3 月 22 日，温家宝总理出席中国发展高层论坛 2010 年会时说：“中国失业人口有 2 亿”。这个数据说明中国的就业问题已经快把“天”给捅破了。

1.1 高校毕业生就业途径

目前高校毕业生就业除了求职找工作外，笔者还观察到以下几个途径：

(1) 在父母等亲属的帮助下就业。这或多或少地涉及社会公平的问题，而且对于大多数人这种途径无法找到理想的工作。

(2) 考研考博，延长在学校中的时间，拖延就业问题。读取更高的学历而成为高端科研人员只能是少数人，否则就不叫高端科研人员，而叫大众科研人员。这种拖延实际上是暂时逃避最终不可回避的就业问题，对于大多数人这种方式不可取。

(3) 考公务员。对于绝大多数人，考公务员基本上难如登天，比买彩票致富更不靠谱。

(4) 出国留学。出国耗资巨大，是大多数家庭所无法承受的，而且在很多情况下是将涉世不久的年轻人扔到万里之外过着放羊式的生活，这存在难以控制的风险。对于大多数人这种方式不可行。

(5) 创业。国家和学校对大学生创业提供了很好的政策甚至资金支持。但在有中国特色的社会主义市场经济环境下，大学毕业生没积累、没经验、没关系，自主创业成功率很低，笔者估计不会超过 5%。因此大学生创业不会解决多少就业问题，不过笔者还是建议有条件的大学毕业生创业，创业无论胜败都可以给人生镀上一层金，有着莫大的好处。

(6) 嫁人。对于女大学生可以嫁人逃避就业问题。不过笔者认为这是把自己人生的幸福完全建立在另外一个人的基础上，风险比较高，嫁人需谨慎。还是自己争取就业来实现生存和发展的风险比较小。嫁人这种逃避就业的方式不能普及。

(7) 下基层。大学毕业生可以响应国家号召，去西部，下基层当村官。其实笔者个人觉得有条件的话回老家当村官还是不错的。不过这种就业观念得不到普及，而且制度上基层比一线城市更不公平，因此大多数毕业生并不愿意下基层。

因此求职找工作将成为大多数大学毕业生解决自身就业问题的主要途径，而且解决就业问题主要靠自己，无法完全依赖其他帮助。

由于中国经济仍处于发展阶段，相对于上千万的大学毕业生、数亿的劳动人口，能提供的就业岗位比较少，人力资源供过于需的问题较为严重，于是形成了巨大的失业人口量，这是至少 10 年内所难以根本解决的；而高校毕业生主要通过人才市场求职找工作的方式实现就业，于是招聘会上形成如图 1-2 所示的火爆场面。

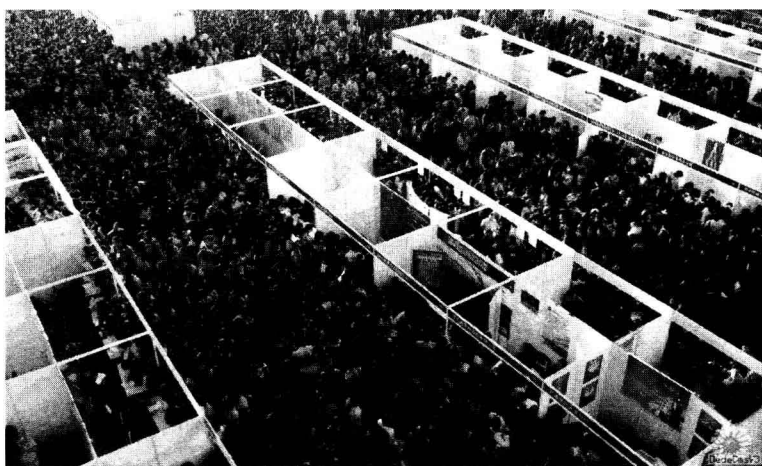


图 1-2 招聘会现场

人力资源较为严重的供大于需，失业人数巨大，这是中国基本国情；目前学校的教育谈不上素质教育，与企业的实际需求有一定的差距，这是中国教育的基本情况。由于历史积累的往届高

校毕业生和应届毕业生的数量太过巨大，因此高校毕业生的就业问题至少在 10 年内是难于解决的。所谓“毕业即失业”的现象将普遍存在，相信在未来很长一段时间内，就业将成为高年级大学生最为焦虑的问题。

根据笔者调研，发现很多软件企业招不到中意的毕业生，存在招到一批毕业生不久又开除一批的现象；而毕业生却在茫茫就业机会中找不到合适的工作，双方都存在矛盾。发生这种情况的原因比较复杂，其中至少一半的原因就是现有的正规学校教育与社会实际需求的脱节，这个矛盾也将长期存在。

在这种大背景下，笔者没有能力系统而完整地解决高校毕业生就业问题，只能对高校毕业生实现就业提供一些通过个人多年从事计算机软件开发职场经验总结而成的指导建议，希望能对高校毕业生顺利就业提供一些帮助。

1.2 计算机专业毕业生的就业问题

这里首先讨论当下比较严重的毕业生就业问题。毕业生就业问题比较严重。

毕业生就业问题也是由于毕业生本身引起的。笔者对南京的软件公司做过一些调查，企业管理者对毕业生最为头疼的不是技术能力，而是其思想意识，特别是其敬业精神和团队合作精神严重不足。尤其是现在开始出现 90 后毕业生，他们中有一些人思想前卫独立，桀骜不驯，以自我为中心，对工作单位是想来就来想走就走，企业管理起来很头疼，这让企业对毕业生用怕了，因此即使招不到合适的人也不大愿意使用唾手可得的毕业生。

出现这种问题，毕业生和企业双方都有原因，本书是面向毕业生的，因此主要说明毕业生如何解决这个问题。

所谓敬业精神就是把工作作为事业来做，而不仅仅是为了赚钱糊口。

一些企业使用大学毕业生大致会有以下步骤：首先是在人才市场和学校中招聘实习生或毕业生，很多毕业生没有实际的能力，而且还需要派老手培训他们，这些都是不小的成本，而且企业还必须为这些毕业生开工资，因此工资不高，也就几百元一个月。

经过一段时期的适应和培训，毕业生们开始具有一定实际的能力了，此时双方产生了利益冲突。

从企业的角度看，由于前期毕业生培训已经投入了大量的成本，此时即使毕业生具有一定的实际生产力，也不大愿意给毕业生开出较高的工资，很可能低于市场行情。

从毕业生的角度看，由于其已经具有一定的能力，具有一定的自信心，此时企业开出的工资低于市场行情，就会导致毕业生跳槽，而且现实中确实有很多毕业生抱着先就业再择业的思想干第一份工作，忠诚度不够，也容易出现跳槽。

这个时候毕业生跳槽，这使得企业将培训毕业生的投入变成为其他企业做嫁衣了，因此企业不大愿意做毕业生培训，而更愿意招聘有工作经验、有实际能力的人；而毕业生没有工作经验就

没有实际能力，这样就形成一个充满矛盾的循环“企业只找有经验的人→毕业生找不到工作→人才市场无法补充有经验的人士→企业招不到合适的人”。

由于存在企业和毕业生的利益之争，就造成企业和毕业生双向选择不搭配的情况。

对此，笔者建议毕业生首先清楚自己的问题，一定要逐渐建立起自己的敬业精神和团队合作精神，这样就能解决就业的主要问题。现在招人的大户是中小民营企业，是有着比较公平的职场竞争环境的，不是“拼爹”，而是拼自己的实力。有能力、有良好职场精神的毕业生一定能很快被企业所发现和重用。

对于这个问题，企业、学校、毕业生三者中，毕业生需要做出最多的努力，因为只有自己才能根本解决自己的问题。指望企业、学校替毕业生解决这些问题是不切实际的。

1.3 求职之路

求职是毕业生职场生涯的开始，对于毕业生来说是人生中一个比较重要的节点。所谓一年之计在于春，有理想、有抱负的毕业生不应该拘泥于先就业再择业的思想，简单地以福利待遇为纲，以眼前的短期利益来抉择求职方向。

比如关于女性择偶有一个段子：在 20 世纪 60 年代，姑娘们都青睐根正苗红的农民，结果 20 年后，农民最辛苦贫穷；20 世纪 70 年代，姑娘们都青睐工人，结果 20 年后工人都下岗了；现在很多姑娘都青睐富二代，可谁能知道 20 年后富二代会是什么样子呢？

笔者建议毕业生们应该把握趋势，结合自己的兴趣和志向，一步步地追寻自己的理想，实现自己的事业。很多杰出人士都是因为有了比较稳定的方向，穷尽一生的时间来专注于社会认可和自己喜爱的事业，才获得成功的。相反，不能确定出长期的事业方向，随波逐流，会很容易成为芸芸众生，淹没在茫茫人海中而无所作为。

1.3.1 抉择

抉择的重要性是大于勤奋努力的，尤其是毕业生在寻找和选择第一份工作的这个关键点时，抉择是非常重要的。正确的抉择需要符合自己的兴趣和志向，此外还得顺应行业趋势。

1.3.2 兴趣和志向

个人兴趣和志向是择业的基础，当工作和个人兴趣相符时，工作就是一件快乐的事情，很容易成功；否则工作就成为一种无奈和折磨，很容易混日子。

有很多毕业生遵循专业对口的原则来进行求职。也就是计算机专业的学生寻找与计算机相关的工作。不过在笔者的经历中，存在大量的“学非所用、用非所学”的情况，大量的非计算机专业的学生以软件开发为职业。

笔者就是动力工程系毕业的，在大学中学习的是热物理、与暖通制冷相关的知识，学习过工

程热力学、传热学、流体力学，不过毕业后一直从事计算机软件开发至今。求职时从未想过专业对口，而笔者的很多同学也存在这种情况。

因此业界大量的实例证明，毕业生寻找工作的时候不必拘泥于所学专业，而应该更多考虑到自己的兴趣和行业的趋势。

每个人的兴趣千变万化，是否对计算机软件开发有兴趣，笔者无法去影响。对于软件开发有兴趣的毕业生，在计算机软件开发行业中，找到一份和个人兴趣相符的工作确实是一件好事。

尊重自己的兴趣，尊重自己的天性，尽量找到符合自己兴趣的职业，这是毕业生在求职时需要认真考虑的方面。关于这方面，每个人的情况不同，应该具体情况具体分析，笔者不在此进行分析了。

1.3.3 顺趋势者昌，逆趋势者亡

行业趋势是任何从业人员都需要考虑和顺应的。无论是行业巨头还是初出茅庐的菜鸟都需要了解趋势，顺应趋势。

趋势是一个很霸道的东西，顺趋势者昌，逆趋势者亡。苹果电脑公司和诺基亚公司就是两个对比强烈的案例。

现在 IT 手机行业有一个明显的智能移动计算趋势，在前几年这个趋势刚刚冒出尚未兴盛的时候，苹果电脑公司和诺基亚公司表现得截然不同，结局也是天壤之别。

图 1-3 是苹果电脑公司 2001—2010 年的股价变化图。

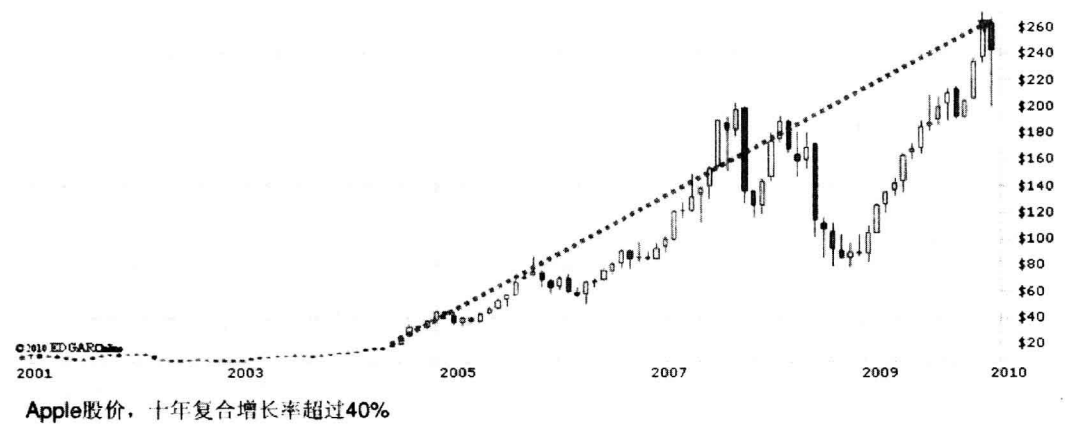


图 1-3 苹果电脑公司 2001—2010 年的股价变化图

图 1-4 是苹果电脑公司年销售额的变化图，单位是百万美元。

从这些数据可以看出，2000 年左右，苹果电脑公司遭遇严重问题，苟延残喘，都快倒闭了。此时苹果电脑公司启用了乔布斯并让其当 CEO，此后苹果电脑公司锐意进取，大胆创新，靠敏锐的觉察力而最终引领了智能移动计算这个行业大趋势，比其他公司早一步推出 iPhone、iPad 等 3G 智能移动数码产品，还创新了配套运营模式。这些顺应趋势并引导趋势的产品都给苹

果电脑公司带来巨大的经济效益。现在苹果电脑公司市值一度达到 3415 亿美元，成为全球最大的 IT 公司，手握现金资产达 762 亿美元。

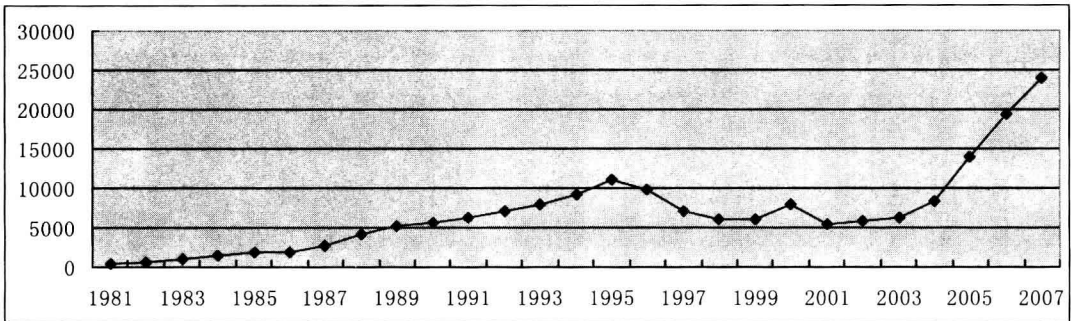


图 1-4 苹果电脑公司年销售额变化图

与此形成鲜明对比的是诺基亚公司。诺基亚公司是 2G 手机时代的霸主，2005 年，诺基亚公司手机发货量达到 2.64 亿部，占全球市场份额的 32%。但诺基亚公司忽视了智能移动计算这个趋势，没能赶上 3G 的潮流，市场份额日趋下降。经过 4 年的时间，诺基亚市值暴跌 87%，业绩大幅下滑。2011 年 8 月 4 日，有研究报表称苹果电脑公司在智能手机的市场份额超过了诺基亚公司，更有报告指出诺基亚手机出货量不仅落后苹果电脑公司，还落后三星公司。市场销售的下滑直接带来了公司的巨额亏损。近期更是出现了中国渠道崩盘的传闻，这对诺基亚公司来说是非常可怕的事情。

诺基亚公司作为老牌科技公司，深知技术非常重要，每年的研发费用高达 40 亿美元。尽管它非常重视具体的技术，但它不识时务，没有发现和掌握趋势，不能顺应从 2G 时代转换为 3G 时代的历史潮流，它正在为自己的迟钝而付出巨大的代价。虽然现在它正在亡羊补牢，靠微软来试图翻盘，但预期还是不好。现在人们从过去的讨论苹果电脑公司什么时候倒闭转变为谈论诺基亚公司什么时候关门。

趋势是如此的重要，因此毕业生要进入软件行业需要对趋势或多或少地了解，然后顺应趋势确定战略方向来寻找工作。

趋势是一种很奇妙的东西，一种连续的变化过程，掌握趋势，需要了解行业的过去，还需要了解行业的现在和可预见的未来，需要感受行业发展的速度、加速度，甚至加速度的加速度。这种知识不是本书所能全面描述的，而是更需要毕业生在未来长期的工作实践中多向外看看，多了解行情，多独立思考，慢慢摸索得到的。不同的人对趋势的把握是不一样的，而客观存在的趋势却是每个人所掌握的趋势的加权统计之合。在此笔者描述一些个人认为的趋势，希望能对毕业生掌握行业趋势有所帮助。

1.4 中国软件行业的特点和趋势

大部分毕业生将在中国国内求职，因此必须了解中国国内的软件行业趋势。

世界软件业界存在着大趋势，而中国存在具有中国特色的软件行业，在世界大趋势下面还有着自己的特点和趋势。

1.4.1 盗版

中国的软件行业深受盗版的影响，这也是中国软件行业最大的特点。

盗版是跨国软件企业对中国的软件产品的超级倾销，盗版严重打击了中国国内的软件企业，使得中国软件行业比较完整地依赖于这些跨国软件企业，这样跨国软件企业就能控制中国软件行业，获得高额利润，而中国软件企业只能可怜巴巴地赚点辛苦钱。关于笔者对盗版的详细说明可参考本书附录 B。

十几年来，中国的软件行业遭受着盗版的毒害，这种情况预计还会持续很久。目前的行业大环境其实不适合技术创新型、产品型软件企业的发展，软件从业者只能先适应这个环境，以后有能力了再改变环境。那么软件开发如何生存和发展呢？

根据笔者的从业经历，推荐几种发展方向。

1.4.2 嵌入式开发

在中国软件行业，嵌入式开发，也就是软硬件结合，才是王道。这个观点是有数据支撑的，表 1-1 是中国工业和信息化部及国家统计局发布的 2012 年中国软件百强名单中的前 20 名。该名单是按照企业软件业务收入进行排名的。

表 1-1 2012 年中国部分软件百强企业名单

序 号	企 业 名 称	软件业务收入 单位：万元
1	华为技术有限公司	8 503 849
2	中兴通讯股份有限公司	3 808 000
3	海尔集团公司	3 024 598
4	北大方正集团有限公司	836 542
5	浪潮集团有限公司	775 126
6	南京联创科技集团股份有限公司	709 286
7	东软集团股份有限公司	646 028
8	浙大网新科技股份有限公司	531 024
9	南京南瑞集团公司（含国电南瑞）	528 783
10	同方股份有限公司	520 641
11	株洲南车时代电气股份有限公司	505 888
12	海信集团有限公司	505 438
13	中国银联股份有限公司	467 433
14	航天信息股份有限公司	446 046
15	神州数码系统集成服务有限公司	442 229

续表

序 号	企 业 名 称	软件业务收入 单位：万元
16	杭州海康威视数字技术股份有限公司	426 302
17	用友软件股份有限公司	410 720
18	福州福大自动化科技有限公司	397 228
19	武汉邮电科学研究院	394 214
20	北京华胜天成科技股份有限公司	354 950

在上面的软件企业名单中，以粗斜体显示的就是典型的软硬件结合的企业。它们都以硬件为主要业务，软件业务成分不高，而且都是作为增值服务的，因此是伪软件企业。

在国内，软硬件结合才是王道，主要原因有：

(1) 国内盗版严重。软件盗版成本很低，但硬件盗版成本比较高。

由于硬件基本上都是以大规模集成电路为核心的，制造门槛高，破解难度大，小打小闹的盗版商是不可能仿造出集成电路的，因此硬件具有很好的知识产权自我保护能力。硬件的知识产权保护能力不仅能保护自己，还能惠顾到软件。这样在软硬件结合中，盗版成本高，盗版现象不算严重。

(2) 国内用户重硬件轻软件，由于硬件看得见摸得着，国内的很多用户愿意花大价钱购买硬件，但就是不大愿意花钱购买软件。这都是盗版把客户惯坏了，一张盗版光盘卖 4 元，这使得人们普遍认为所有的软件都值 4 元钱。因此社会主流严重低估了软件的价值，纯软件价格上不去。但借助软硬件结合，可以将软件的价格转移到硬件的价格上，从而间接地保证了软件的价值，将整个系统卖出去，卖上价。

曾经有这么一个案例，某单位花上百万元进口了一台系统控制设备，是一个大铁盒子。用了一段时间出现故障，但因某种原因无法找到维修人员，于是单位内部的技术人员自己拆开来修，发现这个铁盒子里面是空的，只有一台计算机和几根电缆。原来这个控制设备的价值都是因为其中运行的软件。

中国的软件开发者不大可能改变这个大环境，只能首先适应这个环境，因此嵌入式开发是一个不错的途径。而移动开发也有很多软硬件结合的部分，因此也是一个不错的选择。

1.4.3 移动开发

毋庸置疑，移动开发是一个非常热门的大趋势。移动开发也叫手机开发或移动互联网开发，就是以手机、PDA 等移动便携终端为基础开发软件系统。比如很多人都在用的移动 QQ 就是移动开发的典型。

现在手机已经普及了，人手一台，大部分是 24 小时随身携带的，因此手机的个数和使用次数远远超过 PC，据有关统计，2012 年 1 月，中国手机用户已达到 9.87 亿。

此前移动开发并不火，因为受到手机硬件和通信网络的限制，基础不好，做不出什么好应用。

现在随着智能手机和 3G 网络的快速扩展，突破了此前的限制，移动开发立即进入了一个快速发展通道。据有关统计，2012 年 1 月，3G 手机用户达 1.25 亿，而且增长速度惊人。对于所有的移动开发者来说，每一个 3G 手机用户都是钱罐子，值得深入挖掘。

另外移动开发具有一部分嵌入式开发的成分，也就是部分具有软硬件结合的特性，因此也能适应中国盗版严重的国情。

目前移动开发遇到以下几个挑战：

(1) 大众的移动应用意识不够强。大部分人对手机的应用只限制在打电话发短信的层次，还没有想过将工作生活方面的各种应用通过手机来完成，用户自主提出的需求少。这是一个薄弱点，同时也是移动开发者能发力的地方。

首先移动应用在大众的概念中是比软件应用更神奇的高科技，是一个很好的商业题材，能卖出好价钱。

其次用户自主提出需求少，但能让开发者挖掘出的需求非常多，至少将 PC 应用的功能移植到移动平台上就能挖掘出很多需求。

另外就是移动开发刚刚崛起，市场方兴未艾，而且竞争对手少，能达到比较好的商业收益。

(2) 平台复杂。现在流行着很多软件厂商的移动操作系统，有苹果的、塞班的、微软的、谷歌的，等等。针对某个具体的操作系统做开发比较容易，但应用面窄，而做出能同时运行在几个移动操作系统上的应用则是非常复杂的，开发成本高。

(3) 技术有难度。目前移动开发比 PC 平台的开发要复杂，业界用到的技术还比较少，开发工具还不够成熟，而且移动开发经常需要操作硬件，处理底层网络技术，技术难度大。

不过这些障碍不能阻挡移动开发的发展趋势，识时务者为俊杰，作为开发者应该顺应潮流，关注移动开发。

1.4.4 SAAS 及云计算

虽然在国内纯粹的软件卖不出好价格，但软件开发者还可以在服务上做文章，挖掘商业机会，即不卖软件卖服务。现在互联网业务、云计算、网络游戏等都用这种方式。

SAAS (Software As A Service, 软件即服务) 就是当前软件行业的一个发展趋势，同时也符合中国国情，当下最火的云计算就是 SASS 的一种形式

SAAS 大体上就是用户运行一个客户端软件，通过互联网连接到一个服务器，然后用户使用软件功能，并根据使用量等进行收费，比如按次收费或按月收费等。业界的网络游戏公司出售点卡，腾讯公司卖 Q 币等都用这样的形式。

SAAS 造就了许多企业巨头，如表 1-2 所示的是 2009 年 3 月全球市值最高的 15 家互联网企业排名。

表 1-2 企业排名

排 名	企 业	国 家	市值（亿美元）
1	Google	美国	1023
2	Amazon.com	美国	294
3	Yahoo!	美国	188
4	eBay	美国	153
5	Yahoo! Japan	日本	142
6	Sofitbank	日本	132
7	Activision	美国	129
8	腾讯	中国	120
9	百度	中国	59
10	Rakuten	日本	58
11	Electronics Arts	美国	57
12	阿里巴巴	中国	50
13	NHN	韩国	46
14	网易	中国	28
15	Expedia	美国	21
来源：摩根斯坦利，Economy + Internet TrendsMarch，2009 年 3 月			

这些巨头的成长靠的是海量的用户群数量来获得巨大的利益。表 1-3 是 2009 年年初全球最大的 12 家互联网企业用户数的排名。

表 1-3 用户数排名

排 名	互联网企业	独立访问者（亿）	主要业务
1	Google	7.76	搜索、邮件
2	Microsoft MSN	6.47	搜索、邮件、门户
3	Yahoo!	5.63	搜索、邮件、门户
4	AOL	2.73	搜索、门户、即时通信
5	Wikipedia	2.73	互动百科
6	eBay	2.41	拍卖
7	Facebook	2.22	社交
8	Amazon	1.87	网络零售
9	MySpace	1.73	社交
10	Ask	1.65	搜索
11	腾讯	1.59	门户、即时通信
12	百度	1.52	搜索、社区
注：以上数据不包括在网吧或用手机移动设备的人员			
来源：comScore			

这些互联网企业提醒我们，软件开发者也可以使用 SAAS 模式来适应中国国情，即设计出合

适的商业模式，扎实地做好服务，还是有可能生存和发展下去的。

SAAS 结合移动开发，也就是当下最火爆的互联网智能移动开发，这是值得毕业生特别关注的。

SAAS 具有很强的抵制盗版的能力，也适应中国国情，这是因为 SAAS 提供的不仅仅是一个软件，更是一种服务，而服务是服务提供商实实在在、长时间、不间断地提供的。此时盗版商没有能力提供这种服务，因此也就没人需要这样的盗版软件了。

比如 QQ 聊天软件甚至是免费的，如果离开腾讯的服务器，QQ 客户端软件也就没用了。只要人们使用 QQ 软件连接到腾讯公司控制的服务器，腾讯公司通过发布广告或提供增值服务就能赚钱，而盗版商是无法控制服务器的，也就赚不到钱了。

当然盗版商也能通过架设私服的方式来盗版软件和服务，但这种方式的成本比简单的软件盗版高，而且容易遭到打击。

SAAS 是一个很大的话题，此处不深入探讨下去了。

1.5 行业应用软件开发

行业应用软件开发也是符合中国国情的，也是软件开发者生存和发展之道。

行业软件是软件厂商针对特定行业的用户度身定做的信息化应用软件系统。例如，针对移动通信行业的移动通信缴费系统，针对电力系统的负荷控制系统等。

行业应用软件自身具有一定的抵制盗版的能力，其主要表现在：

(1) 行业软件具有高度定制性，只能为某个用户使用，其他用户不能拿过来直接使用。例如，某软件企业为中国电信开发的缴费系统是不可能用于联通的缴费业务的，甚至为江苏电信开发的缴费系统也不可能直接用于山东电信。

对于行业应用软件，由于各个行业的需求差异和同行业各个具体客户的需求差异形成了一个软件复制使用的壁垒，这加大了软件盗版的成本，因此行业应用软件具有一定的自我保护能力。

(2) 行业应用软件很多是面向企业用户的，企业用户在确定使用一些关系到自身切身利益的应用软件时不大可能使用来路不明的没有售后服务的盗版软件，而是找软件企业为其度身定做应用软件，其中还可能进行招标活动。此时企业用户和软件企业会签订行业应用软件开发合同，合同中规定了软件的基本功能、价格、工期和售后服务等事宜，可能还规定软件的版权归企业用户拥有。

由于行业应用软件有很多是签订合同的项目软件，因此企业用户针对行业应用软件没有盗版的需求，这就大大限制了使用盗版的情况。

由于行业应用软件本身具有一定的对抗盗版的能力，在中国薄弱的知识产权保护的大环境

下，大多数中国软件企业都选择从事行业应用软件的开发，如服务于教育行业、医疗行业、电力行业、制造行业、金融行业等。

大量的中国软件企业从事各种行业应用软件的生产，这就直接推动了中国建设全面的信息化社会。因此软件企业在促进国家和社会进步方面做出了应有的贡献，同时也实现了自身的发展。

1.6 求职战略方向

毕业生求职首先需要确定战略方向，然后采取一定的战术手段。战略方向就是确定工作内容

的方向。

在理想情况下，毕业生首先要很快确定出自己一生的事业方向，因为用穷尽一生的时间来专注于一项自己认可和喜爱的事业，那么成功的可能性就很大了。反之，如果不能确定出长期的事业方向，随波逐流，那么很容易被淹没在茫茫人海中。

对于少数精英毕业生，是可以做到这点的，这基本上靠天赋和运气，不是他人所能学会的；但对于大多数毕业生来说，很难在短期内做出正确的选择。

业界大量案例证明，毕业生寻找工作的时候不必拘泥于所学专业，而应该更多地考虑到自己的兴趣和行业的趋势。

每个人的兴趣千变万化，如果你对软件开发有兴趣，建议还是参与到计算机软件开发行业中，有一份和个人兴趣相符的工作确实是一件幸福的事。

在进入软件开发行业后，毕业生还需要了解行业趋势。因为趋势是最为强大的，顺应趋势，很容易成功；逆势而为，即使是最强大的巨人也会很快倒下，这是一种规律。

1.7 求职战术手段

毕业生在求职的过程中要脱颖而出，除了靠自己的实力外，还需要注意一些战术手段。可以说掌握战术手段本身也算是一种实力。

1.7.1 接触用人企业

毕业生可以通过多种方式来接触用人企业，笔者推荐以下几种。

1. 校园招聘

有很多大中型企业直接进入校园开宣讲会 and 招聘会，这种企业大多有实力。毕业生在定下求职战略方向后就可以在校园招聘会上有选择地递交简历。

2. 人力资源市场双向选择

现在政府很重视高校毕业生的就业问题，为此会组织不少适合高校毕业生参与的招聘会。有时学校会组织大批学生参加这种招聘会，此时毕业生就可以接触大量的用人企业。

3. 网络信息

毕业生可以通过网络访问一些招聘站点来接触用人企业。这些网站包括政府的人力资源部门建设的招聘网站，比如南京的 <http://www.njsrc.com>，毕业生应当关注这些政府背景的网站；此外还有大量的民营招聘网站，比如 51JOB，智联招聘等；还有很多自建门户网站的企业也会发布招聘信息。通过这些网站，毕业生可以获得不少招聘信息。

4. 朋友师兄介绍

毕业生可以通过朋友或师兄介绍来接触用人企业。这些人对用人企业或多或少有所了解。这些了解可能会比用人企业的官方介绍来得更实际，对选择有更好的指导作用，因此有条件就用上。

5. 筛选用人企业

毕业生求职的时候，学业基本完成，毕业设计的工作内容不算多，因此有的是时间用于求职，此时可以采用来者不拒的方式来增加面试的机会，成为“面霸”，有些还升级为“拒无霸”。

笔者建议毕业生对这些用人企业要进行筛选。立志高远、管理规范的企业是很重视员工培训的，毕业生可以询问公司是否有系统的员工培训计划，如果有则优先考虑。

1.7.2 求职过程

人生中第一份工作是比较重要的，因此当毕业生获得一些用人企业信息后需要进行筛选，挑选出管理规范、符合自己特长和兴趣的职业岗位，然后给予重点关注。

1. 争取面试机会

在准备获取某个职务后就可以向企业争取面试机会，一般就是向企业递交简历，然后等待企业发出面试通知。现在毕业生求职竞争比较激烈，因此可以考虑主动向企业发出面试要求。

2. 简历

简历是毕业生向企业介绍自己的第一份材料。有人将简历做得很详细很豪华，甚至有塑料封面，成本较高。笔者觉得无须这样，企业的人力资源专员们收集了成堆的简历，处理每一个简历的时间有限，因此简历写得很详细作用不大，挑重要的写上一两页纸即可。

最后一定要记住在简历中留下详细的联系方式，包括手机号、电子邮箱等。

3. 准备面试

毕业生获得面试机会后就可以准备面试了。对于软件开发技术类职业，主要需要做好以下准备：

(1) 容貌衣着。面试首先要把自己收拾得干净整洁，衣着也要正式规范些。对于大型企业，可能有所要求，对于中小企业来说，一般要求不是很高，但也不要很随便。要注意细节，比如鞋子、发型，当然眼镜的镜片也要干净。

(2) 时间安排。要了解面试的来回路线，估计路上的时间，不要迟到。

(3) 技能展示准备。虽然毕业生没有多少实际工作经验，但也要把自己的一些软件作品展示出来，为此需要准备好软件的运行环境，确定软件展示的预先流程。

(4) 了解企业信息。知己知彼、百战百胜，因此毕业生应该了解要面试的企业的基本信息。首先在互联网上搜索企业的信息，访问其官方网站，了解企业的基本信息。

4. 面试过程

面试有单独面试和集体面试两种方式。单独面试就是求职者一个人面对多个面试官进行交流。集体面试就是多位求职者面对多个面试官来进行交流。

企业如果进行集中面试，那么求职者需要在企业中等待安排面试。在等待的过程中要注意保持安静，不喧哗，此时可以看看企业的情况，比如装修、卫生情况，员工的活动等，也可以静下心来放松紧张的心情。

单独面试时一对多，此时难免紧张，很可能做一些下意识的小动作，比如搅衣角、摸鼻子、左右微晃等。此时毕业生心理素质要好，要会调整心态。最重要的是毕业生要知道求职双方是平等的，尽管毕业生的气场不够，但也有自己的优势。

面试开头多半被要求自我介绍，时间不长，也就三五分钟。为此毕业生应该事先写出定稿并记在心里。自我介绍包括自己的基本情况、学历背景、自己的优缺点、职场抱负等，需要特别突出年轻人不怕困难、充满激情的优势。

在应聘技术类职务时，面试官可能是行业专家，毕业生不懂装懂是肯定会被戳穿的，此时应该实事求是，知道就说，不知道就虚心请教，比如回答“××技术我不大懂，但我能以最快的速度学习掌握并投入实际工作中，相信这用不了多久时间。”

关于求职网络上有大量的文章进行详细介绍和指导，毕业生可以深入研究，掌握一些小技巧，以便提高求职的成功率，本书不做进一步探讨。

第2章

初入职场

2.1 从学生期到职场期的变轨

如图 2-1 所示，一个完整的人生可以划分为多个时期，大体分为儿童期、学生期、职场期和退休期。

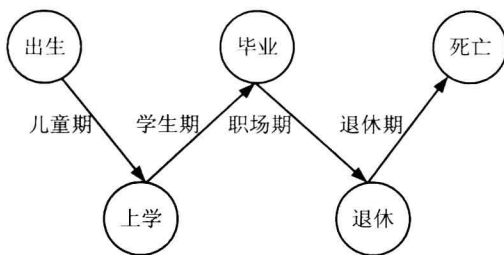


图 2-1 人生的四大时期

人在各个时期都具有自己的生存和发展方式，都具有特定的人生轨道，因此在两个时期之间必然存在转变，存在人生道路的拐弯和变轨。在人生的几次变轨中，以从学生期到职场期的变轨最为重要，也最为剧烈。说重要，那是因为这次变轨会影响人大半辈子的生活，说剧烈是因为这是全方位的变化，控制难度大。毕业生尚未经历过足够的风雨，变轨技巧不够，就像刚学会开车的新手很难搞漂移，搞不好会翻车一样，变轨时可能对后半辈子的人生带来不利影响。

目前社会上存在一种“啃老族”，从学校毕业了，不工作，成天无所事事，完全靠父母供养。毕业生一定要认识到，“啃老族”是失败者，他们就是这次变轨中“翻车”的失败者，这类失败者是大家都瞧不起的，活着是没有尊严的。

金钱、房子之类的身外之物都可以靠父母给，但人生的变轨、思想的转变得完全靠自己。毕业生们初入职场，这是一个过渡期，需要亲身体验这剧烈的变轨。笔者以个人的经验提供一些参考意见，希望能帮助毕业生们顺利变轨，从学校过渡到职场，在人生的道路上来一个完美的转弯，迅速地从毕业生变化成合格的职场人员。

如图 2-2 所示，从学习到职场的变轨是全面的，包括人际交往、角色定位、工作学习方法、

工作目标等，涉及面很广。

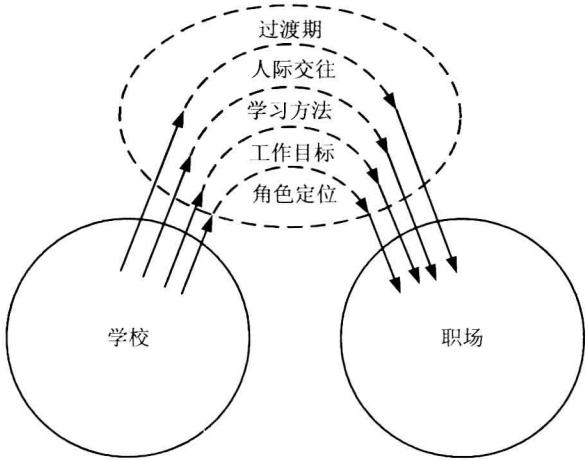


图 2-2 从学习期到职场期的变轨

2.1.1 人生的节日

从另外一个角度看，毕业生开始工作的前 2 年算是人生中一个比较重要的“节日”。
节日按照字面意思可以拆成节字和日字。节就是两段长物体相互连接的部位，比如关节等，日就是日子，表示一天天的时间。因此节日就是连接两段比较平淡的时期的一小段比较特殊的时期。

节日指的是传统的农耕时期传下来的制度，比如立春、夏至、立秋、冬至，还有清明、端阳等。节日并不全都是放假娱乐的，有些是有其重要意义的，指导着农业生产。比如立春就是一年农业生产的开始，此时农民都即刻进入农忙时期，辛勤劳动，下地播下希望的种子，希望金秋能得到好收成。若此时偷懒，错过节气，那么今年的农业丰收就无望了。

毕业生刚刚进入职场的这段时间就是人生的一段大约 2 到 3 年长度的节日，它是连接人生的学生阶段和工作阶段的中间阶段，非常重要，可谓职场中的立春时节。因此毕业生们此时应该辛勤劳动，不急不躁，踏踏实实地打好基础，播下希望的种子，这样就能在几年后得到好的收成，成为职场达人；若此时偷懒，不辛勤劳动，急于求成，则会影响此生的职业发展。

对于职场的立春笔者建议毕业生们需要抱着良好的积极向上的心态度过这段时期，虽然此时有些辛苦，但是播下希望的种子，就像立春时节辛勤劳动的农民一样，心中应该充满希望，充满期待，不应过多地计较其他物质条件，工资待遇等。

2.1.2 团队意识

高校的学习和生活没有多少团队的思想，由于同学们个个追求个性，追求自由，大多是友情关系，较少团队关系，即使存在一些自发形成的学生社团，由于没有强制的制度，也是不大稳

定的。

职场就是战场，本质上就是一个个人加入到以老板为核心的团队中，去对抗另外一个以老板为核心的团队，这种对抗是永久而激烈的，如图 2-3 所示。

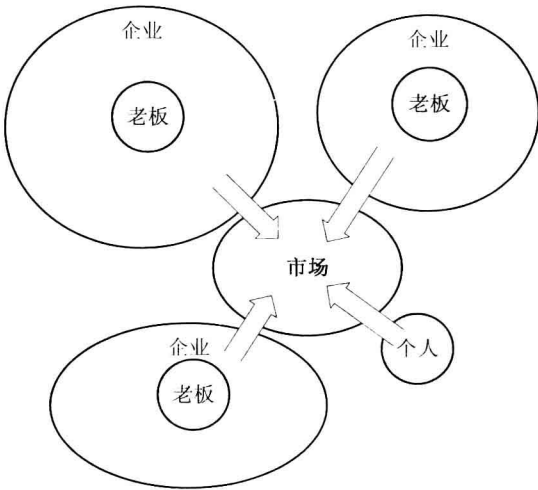


图 2-3 职场的对决

职场是一个以胜败论英雄的战场，讲究的是利益得失，靠的是人多力量大，以大欺小是很正常的，不讲究武侠小说中的公平单挑，玩的就是“群殴”。一个人若不加入团队而冲入职场，很明显会被“群殴”，个人能力再强也会吃亏。就像图 2-4，一个将对付一个以帅为中心的团队，其结局必然是帅生将亡。

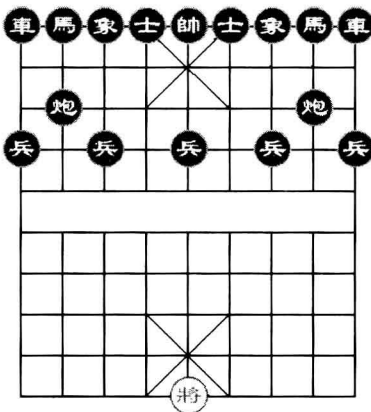


图 2-4 单“将”作战，必败无疑

现在市场竞争激烈，战场上团队和团队之间的空隙很小，难以生存，而且大家又是“群殴”。因此一个人若过于追求个性，追求绝对的自由，没有团队精神，则难以融入任何团队，这是很难生存的。因此毕业生进入职场，要生存，要发展，必须加入某个团队。

笔者认为职场上起码的团队精神大致包括起码的团队忠诚度、团队合作精神和团队工作技能。企业并不指望毕业生能立即带来多大的效益，做出多大的贡献，而是指望毕业生具有一些团队精神来融入团队。但在实践中有些毕业生连这些起码的团队精神都缺失，因此特别强调这些起码的团队精神。



案例：淝水之战

团队中最为典型的的就是军队，因此在说明职场团队之前举一个淝水之战的历史事实来说明团队建设的基础。

东晋时期，五胡乱华，中国分为北方的定都长安的前秦和南方的定都南京的东晋两个政权，两者隔着长江对峙。前秦皇帝苻坚统一北方，踌躇满志，打算南下攻克东晋，此时前秦很多人反对打东晋，指出北方虽然统一，但内部矛盾还比较大，有大量的异族部落武装力量正在隐忍不发。而苻坚却骄狂地说：把我的军队使用的马鞭投入长江中就足以可以让它断流，执意攻打。

公元 383 年 8 月苻坚与其苻融亲率兵 90 万南下，东晋政权没有后路，决意奋起抵抗，派出谢安、谢石、谢玄领 8 万精兵抵御。

战争初期，双方各有胜负，最后双方主力在淝水两岸对峙，谢玄派使者见苻融时用激将法说：“你孤军深入，紧挨着水边摆阵，这不是长久之计。如果你向后移动一段距离，让晋军渡过，然后再决胜负，这不是更好吗？苻坚认为军队稍稍向后移动，晋军渡过一半时再打，就可以取得胜利了。苻融赞同，指挥秦军后撤，但秦军刚打了一个败仗，士气低下，结果几十万大军一撤就失去了控制，阵势大乱。此时谢玄率八千骑兵顺势强渡淝水向秦军猛攻，而秦军中的南方内应此时作乱，大喊：“秦军败了，秦军败了”。秦兵信以为真，争相溃逃，苻融在乱军中被杀，主帅的死亡引起了秦军更大的溃败，从而一发不可收拾。此时晋军乘势追击，收复大量的土地。

苻坚负伤逃回长安，不久被杀，前秦政权分裂。

从这个历史上非常著名的战役中我们可以得出团队建设的经验教训。

对于北方的前秦政权，军队数量占有优势，但它大而不强。

首先它的团队建设不好。内部不团结，存在大量的异族武装力量没有被整合，这些力量非常不忠诚，虽然平时迫于压力忍而不发，但绝对是强大的不稳定因素，最终这些力量导致了前秦政权分裂。

其次是领导者骄狂，做出了错误的决策，带领整个团队走向了错误的方向，这也是很糟糕的。

最后就是执行力不够。前秦军队不能做到令行禁止，主帅让撤退一小段，结果军队组织涣散，一撤退就无法控制，造成大溃败。

对于东晋政权，军队数量不多，但它小而强。

首先它的内部是团结的，从皇帝到百姓都决心一战，心很齐，不会内乱。前方的军队只需全心打仗而无须关心后方的琐事。

其次是领导者很有勇气，做出正确的决策，使得整个团队士气高涨，勇于取胜。

最后就是执行力比较好，东晋的8万精兵经过7年的训练，执行力好，战斗力很强。

前秦的大而不强，东晋的小而强，此消彼长，战争的结局也就很自然了。所谓的用兵如神实际上并不见得高人一等，而仅仅是按照客观科学规律打仗。

从这个战役中我们可以得到，团队建设的三个基础就是团队忠诚度、决策和执行力。决策是团队管理者的事，对于毕业生来说，目前需要重视的就是团队忠诚度和执行力。

1. 起码的团队忠诚度

毕业生初入职场，首先要申请加入一个团队，也就是求职，而只有老板才能批准这个人加入团队。若没有最起码的团队忠诚度，则任何老板是不会批准其加入的，即使偶然加入，也会很快被开除或选择自己离开，产生频繁的跳槽，这当然不好。

大家学过计算机原理，知道在多任务操作系统环境下的 CPU 时间分片的原理。多任务操作系统调度任务不会很频繁，因为每执行一次调度操作，都会消耗掉很多跟执行任务无关的资源。若操作系统没多大事就进行频繁的任务调度，那么这个操作系统的设计是失败的。

同样的道理，一个人跳槽就相当于进行了一次任务调度，期间会花很多时间做善后工作和找新的工作，找工作与人生向前发展是无关的，不能长任何见识，是在浪费时间。每换一个新工作就要重新开始，于是人生就形成锯齿状的高度轨迹，如图 2-5 所示。

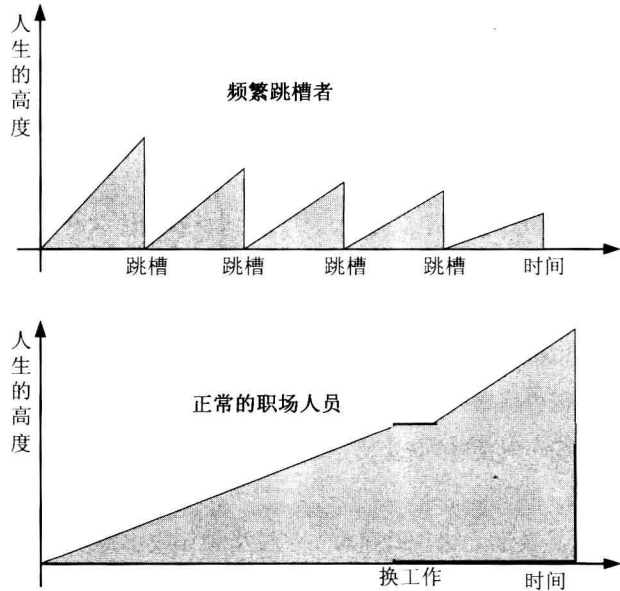


图 2-5 锯齿状的人生轨迹

频繁跳槽者，主观上是想尽快提高自己人生的高度，但客观上却是拔苗助长，得不偿失。频繁跳槽的坏处主要有：

（1）每一次跳槽实际上就是主动放弃了其在目前团队中的地位，也放弃了此前的工作成果，不能保护自己当前工作岗位的投资。

（2）频繁跳槽者会花费大量的时间来进行跳槽，从事本职工作的时间就少了，有效工作经验打了折扣，不容易形成有效积累，使得工作能力提升缓慢。

（3）频繁跳槽会给人浮躁、不负责任的感觉，会影响别人对自己的看法，因此会或多或少地影响自己的人脉关系，给自己带来负面影响。

（4）频繁跳槽能有效地降低人们的自信心，在岁月中空耗了曾经的激情，跳槽在否定目前工作单位的同时也是在怀疑自己的能力，经过频繁的面试和被拒绝，损害了自信心，消磨了激情，这样过几年很容易颓废堕落不可自拔。

（5）有着频繁跳槽的工作简历是非常难看的，一般情况下，老板肯定不会录用频繁跳槽者。求职时隐瞒频繁跳槽经历是掩耳盗铃，因为跳槽导致的较低的工作能力是瞒不住的。

这样经过几年的频繁跳槽，人生的高度只能越来越低。对于长期频繁跳槽而导致的不良结果，笔者是亲眼见过的。因此不知跳槽之弊者，亦不能用跳槽之利。

一个还有正常心态的、具有远见的职场人员是不会频繁跳槽的，而应尽量通过劳动合同自然到期来换工作。长期的工作时间使其更适应工作环境，并获得老板和同事的认可，这样才能更好地进行积累和发展，使其不断攀登人生的高度。

笔者建议毕业生们进入职场时应当具有起码的团队忠诚度，否则在激烈的职场竞争中会被所有的团队抛弃，自己孤独地在一个角落里落后。希望毕业生们既能看到跳槽的好处，也不要忘记跳槽的坏处。

2. 起码的团队合作精神

职场不是小说中的武林世界，而是实实在在的战场。在武林世界中推崇公平的个人单挑，个人能力强或许能获得成功。但在职场推崇的是人多力量大，推崇团队合作，大团队挤压甚至吞并小团队是很正常的事。

毕业生们具备了起码的团队忠诚度后，加入某个团队开始职场生涯，若没有起码的团队合作精神，就不能融入团队，不能与他人一起完成任务。就像一个齿轮箱中有一个齿轮老化会和其他齿轮咬合不紧，迟早是会被替换掉的。

笔者建议毕业生进入职场时要建立起起码的团队合作精神，至少在工作上不要以自我为中心。具体做法可参考以下几点：

（1）搞好和同事的人际关系。这里的同事不仅仅指领导或小集体中与自己有直接来往的同事，还包括更广泛的和自己有间接关系的人。

毕业生刚从学校里出来，还保持着学校中的个性，特立独行，追求自由的风格，一时还改不

了。但企业中有着各色人等，所谓一样米养百样人。毕业生有个性，有些刺头，在学校很宽松的环境中，这没有什么问题。但在企业团队中，很多人在一起做事，此时毕业生如果还在无限制地追求个性，则会引起周围人的不满，从而影响人际关系，不好的人际关系人对己都不好，会影响工作。

笔者建议毕业生要有限制地追求个性，减少刺头，做人不要太冲，不要无故与同事对立，多主动和同事交流，此外还要对同事保持一定的容忍。一堆人挤在一起，难免有些磕磕碰碰，此时应当讲究和谐。团队建设要讲和谐，若少数人不和谐，那么迟早会被“和谐”掉。

(2) 不要建立封闭的小集体。或许有些毕业生和其同校的同学在同一个公司工作，周围陌生的工作环境和比较熟悉的同学关系可能会让这些昔日的同学形成一个封闭的小集体，只在小集体内关系和谐，而对其他人关系一般，外头人进不去，里头的人也不出来。这是比较忌讳的，应当避免。

(3) 少搞八卦，未经证实的消息不传播，不要干涉他人的私生活。侵犯别人的隐私可能会引起别人的戒备。

(4) 不要清高，要虚心学习。清高容易被人孤立，而毕业生刚从象牙塔里出来，存在着清高的倾向，尤其是名牌大学毕业的。此时毕业生需要放下清高，多向别人学习，毕业生刚出来工作，有很多不懂的是正常的，不厌其烦地问别人问题虽然会让人烦，但一般不会有不良影响，还会留下虚心好学的良好印象。

毕业生从事计算机软件开发工作，一般被分配到软件研发部门，软件研发人员一般性格比较内向。在与大家由陌生到熟悉的了解过程中，毕业生可以找自己的师傅或性格比较外向的人交往，然后以此为跳板认识、了解其他人员，广泛开展人际交往。

3. 起码的团队工作技能

毕业生来到一个团队的最终目标就是工作，完成领导交给的任务，而进行人际交往只是完成外围工作，建立一个良好的工作环境，目标就是为了能顺利地开展工作，完成任务。若没有起码的工作技能，就无法完成任务，毕业生来到这个团队就没什么意义了。因此毕业生需要具备起码的团队工作技能。

毕业生由于刚走出学校，因此没有什么工作经验和实际工作技能，对此，企业也是理解的，否则企业就会直接在人才市场上找有工作经验的人了。没有经验和技能不是毕业生不干活的理由，因此毕业生需要立即开始学习，在尽量短的时间内掌握起码的团队工作技能。

从事任何工作都需要其工作人员具备相关的工作技能。各行各业的工作技能是不一样的，但有几点是需要注意的。

(1) 尽量不要给他人带来工作上的不便。现实中一些毕业生虽然完成了某项任务，但完成的结果不好，还需要熟手推倒重来；也有可能毕业生完成了某个工作环节中的任务，但产出质量低下，结果流程跑到下一个人时，由于上一个环节的问题而带来工作不便，需要进行额外的工作。

毕业生需要尽量减少这种情况。

这种情况出现一次两次大家还都能理解，但经常出现，则大家就有意见了。因此毕业生做事应当小心谨慎，多请教同事，工作经验和技能应从现在开始积累。

(2) 发生过的错误尽量少犯或不犯。毕业生刚开始工作，很容易犯错误，这是正常的。但要注意尽量少犯或不犯相同的错误。

(3) 即刻开始积累。每个职场人员都需要积累，那些从业多年、具有丰厚经验和较高工作技能的领导当初也是没什么能力的毕业生，他们的能力都是慢慢积累出来的。因此笔者建议毕业生应从现在开始积累与行业相关的业务知识和工作技能。不要等，不要靠，而要积极主动地学习。

2.2 积累与创新

积累与创新是技术人员良好发展的基础。没有积累，没有创新，软件开发人员就会沦为一个“码奴”，为具体的代码所累，前途堪忧。

其实积累是很多方面所必要的，比如生活中任何人都需要积累生活经验，在工作中需要积累工作经验，而从事软件技术工作，则需要从业者技术上的积累，实现代码方面和思想方面的积累。

代码上的积累就是在工作过程中，将一些用过的通用功能的代码进行整理打包，形成个人的代码库，超越所经历的公司和项目，而且平时有意识地将其完善和发展。这样，今后在开发过程中能直接调用这些平时积累的久经考验的代码库，开发软件就能更有效率。

这个代码库不一定是自己开发的，也可以是他人公布的，但需要花时间了解这些代码，至少要了解其使用及注意事项。现在开源代码比较流行，代码积累的意义有所减弱，但还是需要做的。

其实代码是死的，代码的积累应该说是醉翁之意不完全在酒，代码积累无须在乎代码量的多少，而更在乎其所蕴涵的思想。

软件技术人员的能力分为“代码→技术→思想”三个层次，大量的代码锻炼出技术，技术提炼出思想；反过来思想指导技术，技术创造代码。有点类似读书从薄读到厚，再从厚读到薄的意思。

代码是死的，可能会过时，而思想是活的，永远能发挥作用。不过抽象的思想最终表现为实际的代码而形成生产力，因此代码的积累和思想的积累应该是相辅相成的。

对于一般的毕业生，只有很简单的代码能力，技术能力比较差，思想还谈不上积累，因此需要即刻开始进行技术积累。要阅读高质量的代码，多看书，多向他人学习，同时也需要关注业界同行在如何进行技术开发，逐渐形成自己的技术基础。

一个技术强者仅仅有雄厚的技术基础是不行的，而应当有敏锐的头脑，其智能需要形成一种

金字塔的构造。同时具有雄厚的技术基础和敏锐的头脑，这样才是真正的高手，如图 2-6 所示。



图 2-6 真正的高手具备的素质

敏锐的头脑就是具有强烈的创新精神，创新对企业和个人都具有很重要的现实意义，此处只说明其对个人的意义。

创新是指利用已有的知识基础，通过人脑所产生的联想、直觉等能力创造出前所未有的知识。不过现在的创新强者很多，自己创新的很多新知识说不定别人早就想过了，但这不是重复劳动。首先自己创新的新知识只是很有可能别人已经创新了，但仍然有可能没有被他人创新，因此要对自己有信心。另外创新是一种能力，我们无须跟全球顶尖高手相比，只要跟周围的人相比，自己能创新就是一种很好的优势，养成创新的习惯是非常有意义的，是成为技术高手的必由之路。创新的基础就是具有自己的思想。

有个故事，某国际学校里，老师问学生们：“如何思考其他国家的粮食短缺问题？”，美国孩子问：什么叫其他国家？欧洲孩子问：什么叫短缺？非洲孩子问：什么叫粮食？而中国孩子问：什么叫思考？

中国正规教育制度大部分都要求学生无条件接受书面知识，不需要独立思考。毕业生或多或少地受其影响，独立思考能力比较弱，因此很多人缺乏年轻人所应有的创新精神。这在今后的职场中需要逐渐摆脱正规教育的不良影响，恢复自己的独立思考能力。

创新除了需要有独立思考能力外，还需要一定的危机意识驱动创新。

案例：九城公司

危机和创新是对立统一体。创新可能会避免危机，而不创新则会导致危机；反过来说，危机会导致死亡，但也可能激发创新。在此举几个案例。

第九城市计算机技术咨询（上海）有限公司（简称九城公司）于 2004 年 4 月拿到暴雪公司《魔兽世界》在中国的独家代理权，拥有了 500 万的中国玩家，并于同年 12 月在美国纳斯达克上市。《魔兽世界》贡献了九城公司 3 亿元年收入中的 90%，成为该公司唯一的支柱业务。

2009 年 6 月，网易从九城公司手中抢走《魔兽世界》的代理权。失去了最大的经济支柱，九城公司迅速陷入困境，至今元气未复，CEO 辞职，股价从 50 美元跌倒 6 美元，纳斯达克威

肋摘牌，传闻投资人试图解散公司撤资，当年如日中天的九城公司一夜濒临灭亡。

可以说九城公司小富即安，缺乏危机意识，平时不思进取，不图创新，使其遭到灭顶之灾而束手无策，是一个典型的不创新的反面教材。

另外，创新还需要顺应时代的潮流，要顺势而为，创新才能风生水起，若不顺应大势，硬要拘泥于旧的东西，则会被历史所抛弃，危机四伏。

从业界真实的案例我们可以知道，创新是需要顺应潮流的，创新的方向不对不行，过早或过晚地创新也不行，需要把握创新的时机。那么如何把握创新的时机呢？

这就需要毕业生在埋头学习技术的同时也要关注业界的动态，留心各个热点，多听同行的舆论。在这个过程中掌握相关信息，然后自己思考，最终得出自己的结论。

2.3 技术之害

在此笔者给刚做技术的毕业生一个忠告：“人御技术，技术亦可御人”。笔者对此有着很深刻的体会。



案例：萧远山和慕容博

在《天龙八部》中，萧远山和慕容博潜伏在少林寺偷学 72 绝技，不过这些武学需要和佛法搭配学习才能和谐，结果两个高手眼里只有技术没有佛法，虽然技术学了不少，功力很高，但身体却出现了大问题。这就是人御技术，技术也可反噬人。也可以说，技术对人性有着反弹伤害。

技术的反弹伤害首先导致人性格过于简单，不灵活。

长期专注于技术的人，其性格必然是有点偏的，否则压根就坐不住来研究技术；另外，长期专注于技术也强化了其性格的偏性，说得好听就是执着，说得不好听就是固执。固执就是不圆滑，容易钻牛角尖。

在技术上钻牛角尖是没什么大问题的，因为技术是死的，所遵循的科学规律是固定的，因此比较容易发现是否有出路，若没有出路，则技术人员会退出去。因为技术人员非常尊重科学道理，严格按照逻辑思维，若按照逻辑思维此路不通，则技术人员会尽早退出，不会逆天而行。

但在与社会交往中是没有死规律可遵循的，全是活的，因此自然科学不适用，而长期钻研技术的人其思想已经被技术化，会使用固定的规律处理技术问题，也会使用固定的规律处理社会问题，自然是头破血流。可能有的技术人员不思改变，结果更加封闭自己，更完整地沉入技术，问题会越来越严重。

技术的反弹伤害还造成人以自我为中心，不利生存。

所有的技术都是为人服务的，因此技术人员对待任何技术都是以自我为中心的。若这种技术

思想侵入人性，很容易造成技术人员在社会中也是以自我为中心的。以自我为中心的人活在自己的世界中，有自闭倾向，缺乏与社会的沟通，容易固执，不易接受他人的建议，很难站在他人的角度看待事物。这些都不利于在社会中生存和发展。

以技术讲人术，技术侵蚀人性，这就是技术最大的弊端。人为了更好地发展，就必须获得他人的支持，而已经被技术侵蚀的人际交流能力是人际交往最大的障碍。反过来如果一个人技术好，而且摆脱了技术的限制，人际关系处理得好，处世思路活，那么此人前途无量。因此不知技术之害者不能尽用技术之利。

不过技术是无敌的，即使我们认识到技术之害，技术仍然会慢慢侵蚀人性，这是技术人员无法避免的问题。观察这个商业世界，大多数老板都源自于市场销售部门和财务部门，真正从技术转型的老板比较少。只有很少数的达人能避免技术的这种伤害，而大多数技术人员只能尽量减轻技术的负面伤害。

作为技术初学者，应当注意到这个问题，防微杜渐。技术需要永远地深入钻研下去，需要主动抵制技术侵蚀人性，多与社会交流沟通，保持良好的人际交往关系，这样才能成为复合型人才，这比单纯的技术人员更有前途。越早注意到技术的反弹伤害，技术的反弹伤害就越小。

另外作为老技术人员，应该深刻检查自己的人性，看看技术侵蚀人性到了什么程度，再以无上的精神力量来尽可能地从中消除技术的影响，这个过程是相当痛苦的，简直是不可能完成的任务，不过还得坚持不懈地执行。

2.4 关于薪酬

薪酬是一个敏感的问题，毕业生之间有可能会相互打听薪酬福利，可能会进行比较，薪水低的就会产生失落感，笔者觉得这是没事找事。

实习生或毕业生的工资肯定是不高的，这到哪都一样。这是因为毕业生并没有给公司带来多少经济效益，有时还是一个负担。可能有的同学说现在学费非常高，若毕业后没有足够高的薪酬，那读书不是亏本的生意吗？

其实若按正常情况，23岁毕业，60岁退休，则有37年的工作时间，拿37年的工资。读书是一种长期投资，若自己有能力有耐性把自己做成绩优股，又何必在乎这几年的工资有多少，毕业最初几年工资足够基本生活就可以了。其实笔者觉得各个职场人员的收入差距是在工作5年后开始形成质的差别的，而且收入的差别只是工作能力差别的一种表象而已。

人在不同的人生阶段有特定的主要任务，学生阶段就是学习，工作阶段前期就是赚钱糊口、锻炼能力，工作阶段的后期就是追求事业。而从学校学习到工作之间存在过渡阶段，也就是刚毕业后的几年，还是学习，主要就是学习转变。人生的阶段是一环套一环的，若毕业后不好好经历这几年的过渡阶段，将来的可观收入是空中楼阁，是不现实的。若毕业后几年因为薪酬问题而频繁跳槽，其后果肯定是不好的。

笔者建议毕业生在开始的几年中无须关注薪酬问题，而应关注更重要的角色转变的问题。

2.5 关于买房

关于房价众说纷纭了好多年，一些大学毕业生工作没多少年就考虑买房子的事了，笔者觉得不妥，毕业生买房是弊大于利的。

首先笔者认为房地产疯狂了好多年，不知道还能疯狂多久，关于应不应该买房，众说纷纭，笔者只说说购房费用的事。

现在房价都超过几十万，毕业生刚工作，那么一点点积蓄根本不够塞开发商的牙缝的。于是有两种方式筹集房款，一是向父母要，二是向银行贷款。毕业生都成年了，向父母要一大笔钱并不是什么光彩的事，更要命的就是银行贷款会深重地影响毕业生一辈子的人生轨迹。

比如，某毕业生 23 岁毕业，60 岁退休，工作 37 年。根据人类客观存在的生理特点，其工作激情和年龄的关系大致如图 2-7 所示。

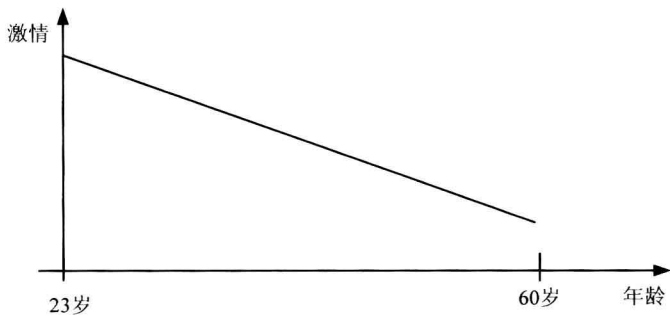


图 2-7 工作激情与年龄的正常关系

该毕业生 25 岁买了 50 万元的房子，首付 20 万元，贷了 20 年期限的 30 万元的房贷。45 岁还清，按照笔者写此文时的银行利息，此人将每月还 1843 元贷款，总共还 442 323 元，其中利息 142 323 元。其工作激情和年龄的关系大致如图 2-8 所示。

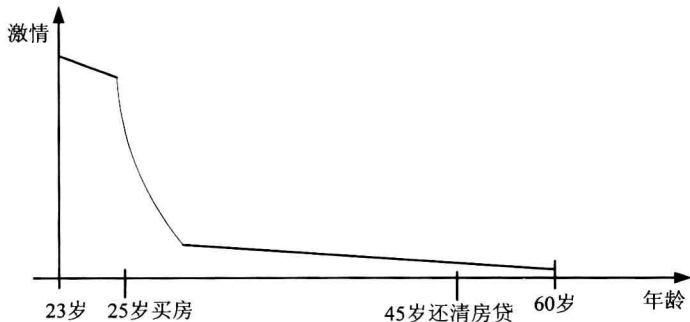


图 2-8 25 岁买房，工作激情与年龄的关系

若该毕业生 35 岁买了 50 万元的房子，首付 20 万元，贷了 10 年期限的 30 万元房贷，45 岁还清，此人每月还 3059 元，总计还 367 191 元，其中利息 67 191 元。其工作激情和年龄的关系大致如图 2-9 所示。

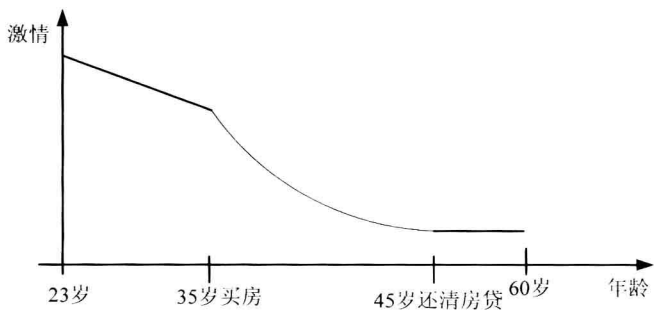


图 2-9 35 岁买房，工作激情与年龄的关系

对比图 2-7 至图 2-9，可以看出买房确实能降低人的激情，因为房贷具有温水煮青蛙的功能，人们在长期房贷债务的压迫下时刻考虑如何稳定地还房贷问题，思想压力大，工作激情受到明显压制，而且为了房贷很可能放弃一些不错的发展机会，从而会对个人事业的发展带来持续性的影响。

根据笔者的理解，事业成功需要很多个人素质的支撑，其中持之以恒的激情是基础素质之一。毕业生若想事业有成就必须一直保持激情，房贷能明显地压制激情，因此立志追求事业的毕业生应当在买房前慎重考虑这点。

通过上述分析可以看到，虽然同是买房，但在 25 岁买房和 35 岁买房的行为对激情的影响存在不小的差别。25 岁买房，激情受到很大压制，一直低位运行；而 35 岁买房，激情受到压制，但不是很厉害，总体上还是中高位运行。这是因为 20 万元的首付对 25 岁和 35 岁的人有着不同的概念。

毕业生 25 岁时没有积蓄，20 万元是天文数字，极其巨大，完全靠东借西凑来获得，其过程比较艰辛，这会比较大地影响人的自信心，而且此时毕业生还不能很好地调整心态，其对人生的激情水平急剧下降而难于恢复。

当人 35 岁时，20 万元也是不小的数字，但不是天文数字，此时事业初具规模，也略有积蓄，说不定能独自承担，至少不用东借西凑；同时收入稳定，能在 10 年中还清贷款，缩短还贷期限，这能少交很多银行利息；而且 35 岁的人已经经历过一些风雨，能调整心态，对人生的激情不会出现明显下滑。

有人要问，现在的房价只涨不跌，买房越迟就越亏。这就事关房地产发展的趋势了，社会上流行很多说法，但谁能预测 10 年以后的房价呢？人们可以预计 10 年以后自己的大致状态，但不能预测 10 年以后的房价，那就先把能预计的事做好，不能预测的事到时候再说，这样做风险比较小。

根据上述分析，笔者建议毕业生不要过早买房，应当量力而行，过早地背负沉重的房债会影响一辈子的工作事业轨迹。

2.6 关于自信心

初入职场的过程就是打碎旧的自信心，重塑新的自信心的过程，破而后立，其过程是有些艰难的。目前社会上的一些不工作靠父母养着的“啃老族”就是没迈过这个坎的失败者，他们已经失去了自信心，颓废没落而不能自拔。

一些学生在学校中由于考试成绩好而得到老师和家长的称赞，同学的羡慕，由此形成以考试成绩为基础的自信心。但毕业后工作了，领导和同事并不在乎考试成绩，若由于高分低能而导致其工作能力不强，被领导和同事批评，这种自信心很快就会丧失掉，人情绪很低落，很容易意志消沉，客观上打破了旧的基于考试成绩的自信心。

此时毕业生需要主观上重新建立基于工作能力的新的自信心，毕业生需要在哪里跌倒了就在哪里爬起来，而不要一直趴在那里不动。这个过程不会很快，有些痛苦，需要坚持，只要能坚持不懈就能慢慢凝练出新的适应职场环境的自信心。

自信心是很重要的东西，没有自信心，一切免谈；有了自信心，一切都有希望。对于毕业生建立适应职场的自信心，笔者建议以下几点。

(1) 一直都要保持对自己最基本的自信。这个最基本的自信就是星星之火，不能熄灭。毕业生需要知道自己是独立的人，是和领导同事一样的人，人与人之间具有现实能力的差别，但都具有无限的潜力。

(2) 保持积极的心态。遇到挫折需要保持积极的心态，振作精神。

(3) 对于成功需要自己称赞自己。毕业生虽然能力不足，但做事总有成功的时候，此时不必指望别人称赞自己，而要自己称赞自己，逐渐形成新的自信心。

2.7 小结

毕业生从学校到职场，这是一个重要的人生道路的变轨。这个变轨很剧烈，需要在很多方面主动地做出变革。此时需要认清主要问题，不要拘泥于次要问题。凝神聚力地实现顺利变轨，从而给几十年的职场之路开个好头。

每个人的职场之路都有漫漫几十年，每个人都追求成功，但所能做出的成就差别巨大。成功有多方面的因素，其中非工作技能的因素非常重要，本书是偏向技术方面的，非技术的就不再深入探讨下去了。

商业软件开发基础

3.1 学习型软件开发和商业软件开发

毕业生在学校可能开发过一些计算机软件，但这是学习或研究性质的软件开发，实际上更偏向是单纯的写代码；而毕业工作了，在软件公司里所进行的软件开发是商业软件开发。学习型软件开发和商业软件开发有比较大的差别，如表 3-1 所示。毕业生必须搞清楚这两者之间的差别，否则工作时会吃力不讨好。

表 3-1 学习型软件开发和商业软件开发的比较

比 较 项 目	学习型软件开发	商业软件开发
最终目的	学习和探索软件开发技能，能通过考试	做出能卖出去的软件产品
成本	不计成本	需要斤斤计较
过程	有所控制	需求调研，计划，审核，开发，测试，部署。一个都不能少
代码，文档规范	没有要求	必须控制
软件质量	稍微有点	必须控制，需要经过测试来达到所要求的软件质量，但不追求过高的质量
兼容性	没有要求	需要控制，能适应特定的软件运行环境
性能	没什么控制	必须有性能，不能低也不能高
用户界面	讲究不多	重视，需要依赖专业的美工人员
时间进度	有所要求	存在明确的时间节点
产品化	没有要求	可能需要用户手册、学习资料、宣传资料、产品包装、产品网站等

现针对这些比较项目深入地探讨下去。

3.2 项目软件和产品软件

商业软件一般分为合同软件和产品软件两类，此外还有一些自营软件。

3.2.1 合同软件

某个行业的客户委托软件厂商开发一套该客户自己使用的定制软件。这种软件功能是定制的，只有这个客户能直接使用，其他单位，即使是同行业也不能直接使用。这种定制软件只针对这个客户自身的特点。由于这种软件开发前都需要签订开发合同，所以称为合同软件。例如中国移动的缴费系统就是项目软件，只能中国移动使用，其他电信运营商用不了。

项目软件具有以下特点：

- (1) 客户是软件开发工作的发起者，可能会进行项目招标。
- (2) 客户和软件开发商签订开发合同，指明软件功能、开发时间和金额。
- (3) 开发出来的软件只能由该客户使用，一般不得给其他客户使用，实际上其他客户也用不了。
- (4) 软件的知识产权一般归客户拥有。
- (5) 软件厂商负责软件的研发，此外还负责软件的部署、客户培训、运行维护等工作。
- (6) 软件开发商可能顺带负责客户的信息化基础建设，如采购硬件和系统软件等。
- (7) 软件部署后大多修改不大，若客户出现大的需求变化则可能会购买新的合同软件。

3.2.2 产品软件

软件厂商针对特定的用户群开发一个通用的软件产品，特定用户群中的每个客户都能直接使用这个软件而无须定制，这就是产品软件。例如 Windows 操作系统、金山毒霸等就是产品软件。

产品软件的特点有：

- (1) 由软件厂商自发进行，不会签订项目软件开发合同。
- (2) 软件功能需求由软件厂商自行收集。
- (3) 开发出来的软件针对某个用户群，而不仅针对单个客户。
- (4) 软件产品需要长期规划，可能需要升级换代。
- (5) 软件知识产权归软件厂商拥有。
- (6) 有软件商业包装，提供完备的用户手册、各种学习资料及产品网站等。

合同软件和产品软件的对比如表 3-2 所示。

表 3-2 合同软件和产品软件的区别

比 较 项 目	合 同 软 件	产 品 软 件
发起者	客户	软件厂商
签订合同	签订	一般不签订

续表

比 较 项 目	合 同 软 件	产 品 软 件
知识产权	一般归客户拥有	软件厂商拥有
软件应用范围	单个客户	用户群体
软件功能	定制	通用
软件包装	一般，仅包括用户手册	好，包括用户手册、各种学习资料、产品网站等
软件技术	够用就行	要求高
软件发展	部署后修改不大	需要升级换代

本书的商业软件开发只讨论合同软件开发，不考虑产品软件开发。

3.2.3 自营软件

软件厂商自己开发和运营一套软件体系，可能软件本身是免费的，但使用过程中会创造价值，给软件厂商带来利益，这种软件叫做自营软件，如腾讯公司的 QQ 聊天软件。自营软件已经不仅仅是软件研发了，更多的是商业运营，本书不讨论自营软件。

3.3 商业软件开发基本概念

学习型软件开发是以软件开发人员为中心的，而商业软件开发是以客户为中心的，这是两者的本质区别。

学习型软件开发是完全由开发人员随心所欲地控制的，只是自发地学习和实践一些软件开发技巧，没有经济目的。毕业生以前的学习型软件开发一般就是编写一些程序代码，算是花拳绣腿，练练腿脚。

商业软件开发是由客户发起的，由客户付费雇佣软件开发商开发软件。顾客是上帝，这个普遍存在的规律使得开发人员不能随心所欲地开发软件。

商业软件开发的主要目标是经济目标，即开发出能赚取利润的软件。软件开发公司虽然视客户为上帝，但以追求自身利润最大化作为终极目标，而软件开发过程只有成本没有收入，为了追求利润，软件公司自然会千方百计地降低软件开发过程中的成本。

3.4 商业软件开发的平衡点

在商业软件开发中存在一对根本矛盾：一方面，客户希望用最便宜的价格购买质量最好、功能最多的软件；另一方面，软件厂商希望以最贵的价格卖出成本最低、功能最少的软件。这对矛盾之间形成一个平衡点，如图 3-1 所示。

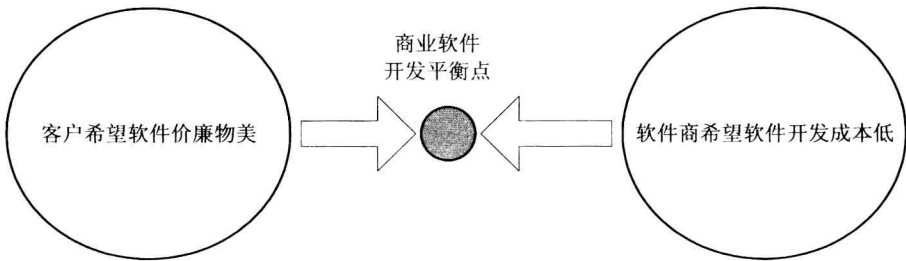


图 3-1 商业软件开发平衡点之一

商业软件开发应当准确地命中这个平衡点，一切围绕满足客户需求并降低开发成本而开展工作。如图 3-2 所示，该平衡点还包括软件开发时间、软件质量和软件功能三方面内容。

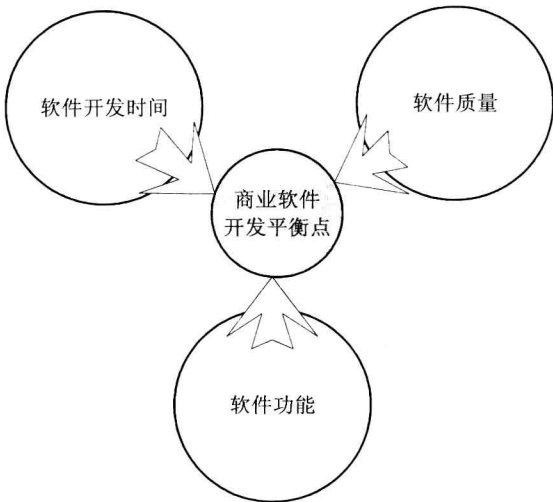


图 3-2 商业软件开发平衡点之二

3.4.1 软件开发时间

商业软件开发时间大多是有明确限制的，在客户和软件公司签订的项目软件开发合同中肯定会明确规定软件的开发完成时间，若开发延期，则软件公司是会赔钱的，因此软件开发时间是固定的，而且大多数是很紧张的。软件开发时间越短，工作强度越大，软件开发整体成本就会越高，但是由于合同的约束，在软件开发时间方面，软件开发厂商是没什么好发挥的。

3.4.2 软件质量

客户希望软件的质量越高越好，软件稳定不出错。但是开发高质量的、运行稳定可靠的软件，其成本是很高的，因此软件厂商并不会开发出很高质量的软件，而是根据客户所实际需要的软件质量来确定所开发的软件质量的，开发出来的商业软件质量会比客户所需要的高一些，但不

会高很多。商业软件开发人员对软件质量的追求是适可而止的。

即便如此，商业软件的质量还是比学习型软件的质量高得多。因为商业软件要在客户那里长期运行，这期间不能出现重要业务数据丢失的情况，因此对很多商业软件的第一要求是安全稳定。一般情况下，商业软件开发要求稳定压倒一切，其次才是软件功能。

3.4.3 软件功能

商业软件应当能实现客户提出的需求，若可能的话可以略有增加，商业软件的功能不会大大超出客户提出的需求。商业软件开发人员对软件功能的追求也是适可而止的，够用就行。

有些商业软件可能存在明确的后续开发需求，比如软件项目分为一期、二期等。此时软件厂商决定软件功能时需要进行额外的考虑。

从这个平衡点的分析上看，在商业软件的开发过程中需要一切以客户的需求为出发点，一切围绕着客户的需求。因此客户需求是商业软件开发的最重要的基础。新手们需要时刻牢记这点，做任何商业软件开发之前都需要把握客户需求。

以商业软件开发平衡点出发，下面介绍一些商业开发的其他方面。

3.5 过程和质量控制

有好的过程不一定有好的结果，但没有好的过程肯定没有好的结果。学习型软件开发是随心所欲的，但商业软件开发需要过程控制。商业软件开发大体经过如表 3-3 所示的步骤。

表 3-3 商业软件开发步骤

步 骤	工 作 内 容	产 生 结 果
需求调研	调研人员去客户处询问、记录、整理用户需求，并将客户原始需求转化为软件功能需求	需求说明书
系统设计	软件开发人员针对软件功能需求进行系统整体设计，包括模块技术，预计采用的技术和开发环境等	系统设计说明书
软件设计	深入进行软件各个模块的详细设计	系统详细设计说明书
软件开发	若干个软件开发人员一起完成软件功能的开发	程序文件等
软件测试	测试人员对照需求说明书进行软件的功能测试	测试结果文档
产品化	编写用户手册、学习资料等	用户手册等
部署	软件在客户现场进行部署	工作日志
维护	软件系统的维护，对其中的错误进行修改	工作日志

学习型软件开发的需求调研和软件设计是草草而过的，软件开发后就结束了；而商业软件开发则是严谨有步骤的。这几个步骤都是业界经过长期商业软件开发的实践而制订出来的。一些软件开发认证体系，比如 CMMI 等更是将这些步骤进一步制度化、系统化。

商业软件开发的过程需要进行质量控制，没有过程和质量控制的软件开发过程很有可能成为开发人员的血泪史。

关于过程和质量控制将在后面的章节详细介绍。

3.6 新旧技术的权衡

现在软件业界的新技术越来越多，商业软件开发中可以采用一些新技术来与时俱进，但需要经过权衡，进行利弊分析。

拒绝一切新技术是没有前途的，因为用户需求越来越复杂，旧技术可能很难或无法实现新的用户需求。此时软件厂商由于其产品老化，功能不足，成本越来越高而很容易被市场淘汰。

盲目采用新技术是很冒险的行为。采用新技术可能会带来以下几个问题：

（1）新技术自身没有经过长期的实践检验，可能存在问题，这会影响到商业软件的安全和稳定。

（2）新技术可能带来兼容性问题。开发人员以前掌握的开发技巧，旧的正在运行的软件模块都可能和新技术发生冲突。

（3）新技术带来学习成本。开发人员需要花时间、花精力学习新技术，这会影响到商业软件开发的时间进度。

（4）新技术可能不实用。新技术在投入使用后可能毫无作用或起负面作用。这样，此前对新技术的投入毫无效果，且加大商业软件的开发成本。

采用新技术在短期内必然会导致软件开发成本上升，而且长期也不一定会降低开发成本。因此需要软件开发厂商在使用新技术前进行权衡和利弊分析。新技术是为了有效地降低长期成本，而不是软件开发人员个人的技术镀金。

3.7 商业包装

商业软件属于商业产品，基本上所有产品都需要商业包装，否则很难卖上价。

案例：小品《如此包装》

春晚上曾经演过巩汉林和赵丽蓉合作的名为《如此包装》的小品，里面就涉及了商业产品的概念、名称、用户体验和系统化。图 3-3 是该小品视频的一个截屏。

在该小品中，赵丽蓉被换了一个英文名，用英文打招呼，那就是在进行产品概念、名称的包

装。著名的“嗨，嗯哼”就源于这个小品。



图 3-3 《如此包装》小品视频截屏 1

赵丽蓉换服装，换背景音乐，那就是改善产品的用户体验，使其符合大众口味。

赵丽蓉不是一个人在唱评剧，后面还跟着一群美女伴舞，那就是进行产品的系统化，如图 3-4 所示。



图 3-4 《如此包装》小品视频截频 2

学习型软件研发没有商业包装，开发人员写代码编译生成一个程序文件就算完事。

商业软件需要商业包装。软件名称大多是在合同中确定的，比如“某公司办公管理系统”之类的，这个软件厂商可以宣布软件采用了某软件巨头的先进的×× workflow 技术，并自主实现了×× 技术等。

商业软件的用户界面需要经过专业美工的美化，使其符合大众口味。

商业软件不是一个单独的程序文件，还有用户手册、学习视频、客户培训等配套资料，以便进行产品的系统化。

3.8 商业软件开发人员的工作环境

毕业生需要了解商业软件开发者在软件公司中的位置，埋头苦干的精神是要发扬的，但了解工作环境也有利于更好地工作。这里的工作环境不仅仅指工作的场所，还包括公司的组织架构，人员的分工等情况。

如图 3-5 所示，在一个软件公司中，主要包括管理层、软件开发人员和市场营销人员。

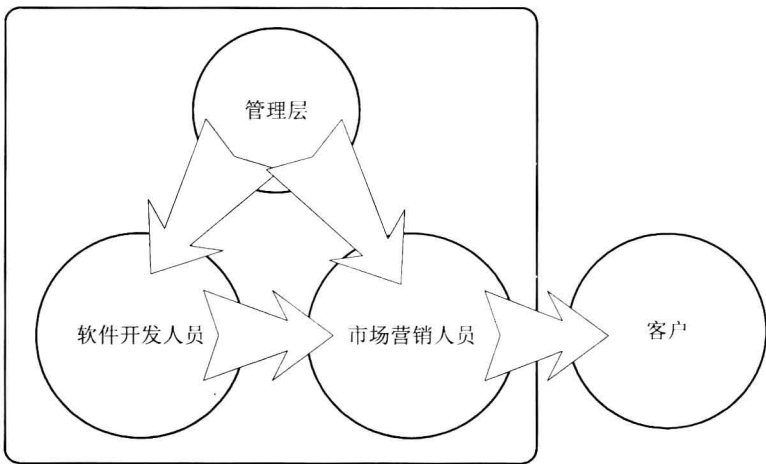


图 3-5 软件公司架构

管理层包括企业老板、各级经理等，它对软件开发人员和市场营销人员下达工作指令，是一个软件公司中最优势的人群，同时也是责任最大的人群，所有的重大责任都可归到管理层。

市场营销人员负责市场营销工作，跑客户，拿合同，是软件公司直接的创收人群。

软件开发人员负责软件项目开发、部署和维护，是软件公司的技术驱动力，也是成本大户。

在公司之外的就是客户，客户接受公司提供的软件产品和服务，并为这些向公司支付费用。

以下从软件开发人员的角度对工作环境进行说明。

3.8.1 对于客户

只有客户为软件公司提供的软件产品和服务支付费用时，软件公司才有利润，才有存在的意义。因此软件公司的一切工作都直接或间接地围绕着客户来展开。毕业生需要时刻记着要开发精准的符合客户需求的软件。

软件开发人员服务的客户有两种：一种是在公司内部替客户开发软件的；另一种是在客户现场帮助客户进行软件项目的实施和维护的。

在公司内部替客户开发软件时需要充分理解客户需求。商业软件是软件开发人员对于客户需

求的一种描述，不理解客户需求，技术再好也是南辕北辙。因此商业软件开发的第一步就是准确地获得和理解客户需求。

在客户现场帮助客户进行软件项目的实施和维护时应注意，这是帮助客户而不是代替客户。因为软件最终是为客户所用的，软件开发人员不可能替客户使用这个软件，但客户不懂技术，因此软件开发人员需要帮助客户部署和维护软件，还需要提醒和帮助客户创建运行软件的各种必备条件。

3.8.2 对于市场销售人员

若市场销售人员是冲锋陷阵的士兵，那么软件开发人员就是兵工厂，两者都是不可缺少的。不过在大多数情况下，市场销售人员比软件开发人员具有优势，这是中国的基本国情所决定的。

中国的市场经济体制还不够成熟，商业交易规则很复杂，交易成本高，难以控制；但中国人力资源丰富，软件技术人员比较多，而且软件技术开发规则相对于商业交易规则是简单的，比较好控制。这就迫使管理层将更多的精力放在市场营销工作上，因此市场营销人员在公司中是占有优势的，这点在小型公司更突出。

不过对于有志于软件开发工作的毕业生也无须多虑，因为优势和劣势是对立统一的。市场营销工作也是很复杂的，国内的交易规则很复杂，做好市场工作不容易，需要付出不小的代价。软件开发工作由于规则简单，认真做还是能做好的，而且随着国内软件行业的不断进步，软件开发人员的重要性是会日益突出的。

认清软件开发人员在软件公司中的位置，就能更有针对性地开展工作了。

软件开发人员性格内向的比较多，人际交往能力不强，以自我为中心，不太能了解市场人员的处境，反而觉得市场人员成天吃吃喝喝，到处忽悠，有点瞧不起。笔者建议软件开发人员要放下所谓的清高，多和市场人员交流，掌握客户更多的信息，这有助于开发更符合客户需求的软件，而且能降低公司管理的复杂性。市场交易规则已经很复杂了，若一个软件公司内部的管理规则也很复杂，那么就很容易淹没在由几万家软件公司组成的汪洋大海中了。

软件开发人员对市场销售人员还需要普及一些基本的技术概念，以避免市场销售人员为了争夺合同而对客户做出不切实际的技术承诺。这些不切实际的技术承诺是个苦差事，对大家都没有好处，但市场销售人员不精于技术，当初可能不了解，对此软件开发人员需要利用自己的优势主动预防这种事。

3.8.3 对于管理层

软件开发人员对于公司管理层，也就是各级领导，也要尊重、服从，外加利用。领导中有很多是软件行业的老前辈，尊老是中国的老传统，领导是上级，服从上级是应该的。利用领导分为两种：一种是利用领导的经验和智慧，另一种是利用领导的权力。

毕业生需要利用领导的经验和智慧。毕业生作为新手，经验尚未积累，能力有限，因此在工

作时会遇到各种困难和难题，独立思考和解决固然很好，但那是要花时间的，在学校有充分的思考时间，但在企业中时间就是金钱，是没有那么多时间独立思考的，此时需要向领导和同事请教，充分利用领导的经验和智慧。

毕业生不需要顾虑所问的问题是否很简单、很可笑，其实大家都了解毕业生的状态，是不会这么看的。但是问问题时应当找直接领导，一般不要越级。

毕业生需要利用领导的权力，在工作中遇到一些不能自己决定的事情时，尤其是在客户处，要及时地请示领导，让领导做出决定，这种事不能拖，若找不到直接领导，那么就找间接领导。

第 4 章

开发者眼里的 Windows

此前的章节都在介绍毕业生进入软件开发行业中应该有的一些思想准备，从本章起笔者开始带领毕业生深入技术领域，掌握在商业软件中用到的技术知识。

微软 Windows 操作系统到处都在运行。相对于非专业的普通大众而言，开发者眼里的 Windows 操作系统内容丰富多。

下面介绍一些 Windows 知识，这些知识普通大众是无须知道的，但专业的开发者却是必须掌握的。

4.1 Windows Service

Windows Service，也称 Windows 服务，是 Windows 操作系统中一种长期运行的后台程序。它们长期在后台运行，没有用户界面，默默无闻，但它们却是支持 Windows 正常运行的幕后英雄，永无出头之日。

Windows 服务程序为其他系统模块提供了非常重要的服务，而且各个 Windows 服务分工明确，比如有个名为“Print Spooler”的服务用于提供打印支持，若该服务不启动，则任何软件都不能进行打印，比如 Word、记事本或报表软件等。

Windows 操作系统启动后还没等用户登录就会启动一些 Windows Service。Windows NT 和 Windows 2000，以及更新的版本都能运行 Windows Service，但 Windows 98 及其前期版本是不支持 Windows Service 的。

对于开发人员来说，比较重要的 Windows Service 有：

- 与 IIS 相关的服务，包括基础的 IISAdmin 服务和 W3SVC（World Wide Web Publishing）服务，这些服务提供在网络上发布 Web 内容的功能。这是 ASP.NET 应用程序运行的基础，如果这些服务没启动，则静态 HTML 页面、ASP、ASP.NET 或 Web Service 等都不能运行。
- 与数据库管理系统相关的服务，很多运行在 Windows 系统中的大型数据库系统，比如 MS SQL Server、Oracle 等，其数据库管理系统都是以 Windows Service 模式运行的。比如对于 MS SQL Server 2000，其核心程序的服务名称为 MSSQLSERVER。若这些服务没有启动，则数据库是绝对连不上的。

4.2 管理 Windows Service

打开 Windows 资源管理器，在左边的树状列表中选中“桌面→控制面板→管理工具”，如图 4-1 所示。

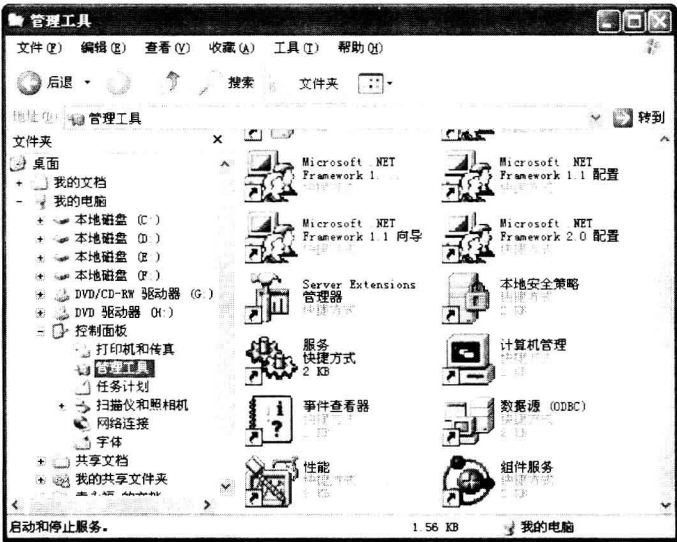


图 4-1 选中“管理工具”

在右边的列表中打开“服务”项目即可打开 Windows 服务管理器，如图 4-2 所示。

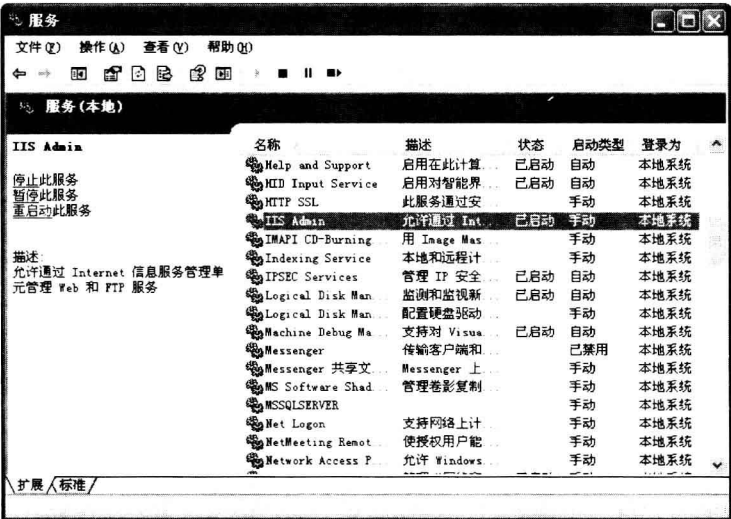


图 4-2 Windows 服务管理器

在这些服务中，让开发人员最牵肠挂肚的有 IIS Admin 和 World Wide Web Publishing 服务。在这个列表中双击一个服务项目即可打开如图 4-3 所示的服务属性对话框。

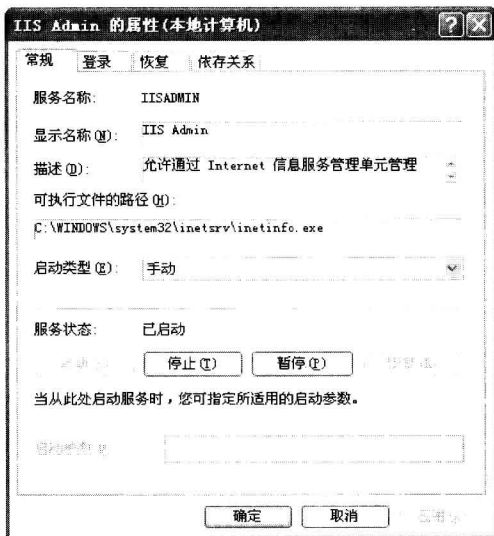


图 4-3 服务属性对话框

Windows 服务有一个服务名称属性，该属性是服务的唯一不可重复的名称，我们可以在命令行中使用命令“net start 服务名称”来启动服务，使用“net stop 服务名称”来停止服务。

Windows 服务的启动类型有自动、手动和已禁用 3 种。当启动类型为自动时，Windows 启动后不等用户登录就自动启动服务；当启动类型为手动时，需要某个操作员登录后单击“启动”按钮来启动服务；而当启动类型为已禁用时，Windows 服务不能启动。

该页面中的“启动”按钮用于启动尚未启动的 Windows 服务，运行提供服务的进程；“停止”按钮用于停止已经启动的服务，杀死服务进程；“暂停”按钮用于通知服务进程暂时停止提供服务，但服务进程依然存在；“恢复”按钮用于通知处于暂停模式的服务进程重新提供服务。

进入服务属性对话框中的“登录”选项卡，如图 4-4 所示。可以指定服务使用本地系统账户登录，也可以另外指定其他用户，这里有一个“允许服务与桌面交互”的选项，若选中此选项，则 Windows 服务可以显示图形化用户界面，比如显示自己的窗体，显示消息框等。不建议使用该选项，而且 Windows 服务运行时不要显示图形化用户界面。

切换到“依存关系”选项卡，如图 4-5 所示，可以看到本服务和其他服务的依存关系。

各个 Windows 服务之间可能存在依赖关系，比如 IIS Admin 服务就依赖另外一个名为 RPC 的 Windows 服务，当启动一个 Windows 服务时，系统会启动该服务所依赖的其他 Windows 服务。例如，我们设置 IIS Admin 服务为自动启动，而 RPC 服务为手动启动，则 Windows 启动后会试图自动启动 IIS Admin 服务，即使 RPC 服务不是自动启动的，也会首先启动 RPC 服务。若 RPC 服务为禁止，无论如何也不能启动，则 IIS Admin 服务就无法自动启动了。

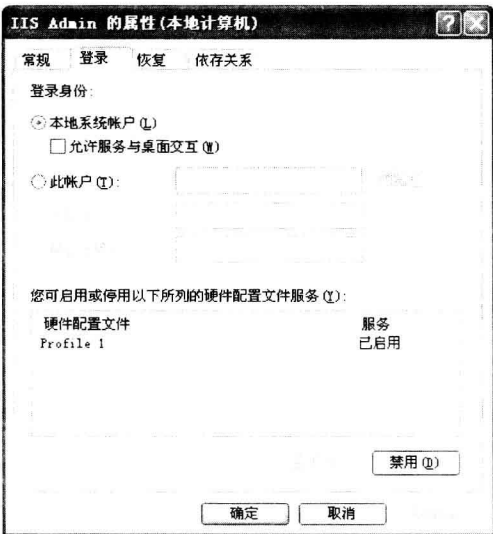


图 4-4 “登录”选项卡

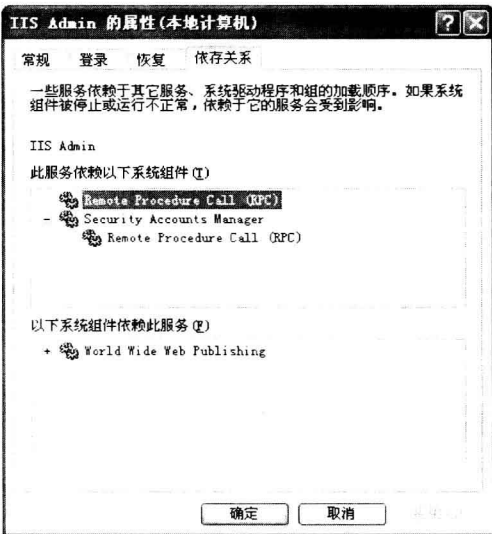


图 4-5 “依存关系”选项卡

4.3 事件查看器

事件查看器能查看 Windows 系统和一些程序产生的 Windows 系统级重要日志信息。一般来说，系统级日志信息是比较重要的，是了解 Windows 操作系统和重要系统组件的重要手段。

如图 4-6 所示，在 Windows 控制面板的管理工具中双击“事件查看器”就可以启动 Windows 事件查看器。



图 4-6 启动 Windows 事件查看器

Windows XP 的事件查看器的用户界面如图 4-7 所示，其他版本的 Windows 事件查看器的用户界面与此界面有所不同，不过大同小异。

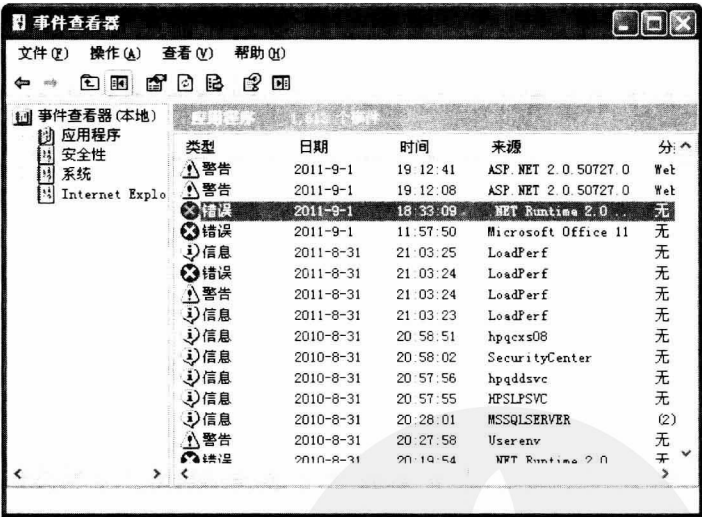


图 4-7 事件查看器的用户界面

在事件查看器的用户界面中，左边是事件类型，右边列出了所有的事件。在右边的事件列表中双击某个事件，就会显示如图 4-8 所示的对话框来显示事件的详细内容。

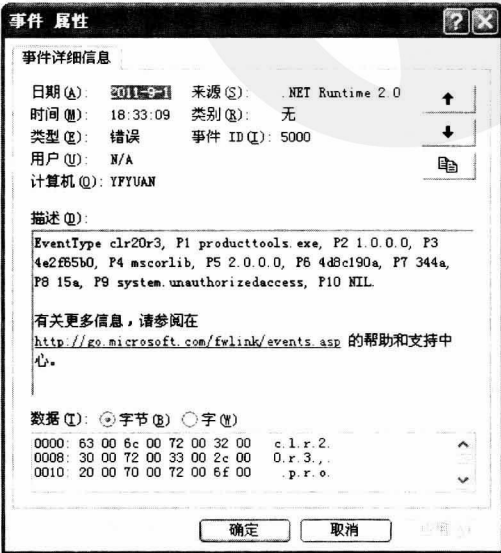


图 4-8 “事件 属性”对话框

可以根据这个事件日志信息来了解，记录这个事件的时刻 Windows 操作系统到底出现了什

么问题。这是调试应用系统的重要手段。

在查看日志时，当觉得内容太多，或者准备重新运行一次应用系统然后再看系统生成什么事件记录时，可以将事件日志清空，操作很简单，如图 4-9 所示。

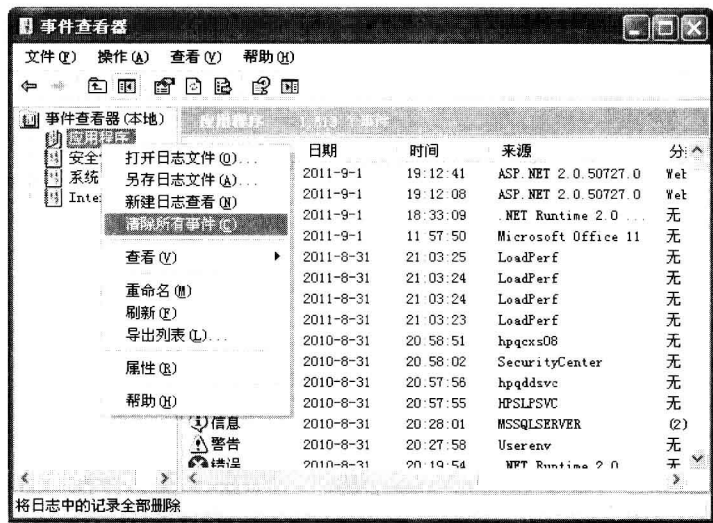


图 4-9 清除所有事件

在图 4-9 右边的列表中用鼠标右键单击某个事件类型节点，弹出快捷菜单，单击其中的“清除所有事件”菜单项目，即可清空某种类型的日志。

4.4 远程桌面

远程桌面类似 QQ 的远程协助功能，能查看和操作远程计算机中的资源，其用户界面和操作过程与本地计算机基本一样。在实践中，远程桌面是开发人员和系统管理人员操作服务器的重要途径，可以坐在舒适的办公室中操作服务器，而不必跑到嘈杂的机房中站在服务器面前操作。

4.4.1 配置服务器

首先用管理员账号在服务器中进行本地登录（站在服务器前面现场登录），打开“系统属性”对话框，切换到“远程”选项卡，如图 4-10 所示。

勾选“允许用户远程连接到此计算机”，注意能远程连接的用户必须设置密码，一些操作系统允许用户设置空密码来实现自动登录，而空密码的用户是不能接受远程桌面连接的。

单击“选择远程用户”按钮，弹出如图 4-11 所示的对话框。

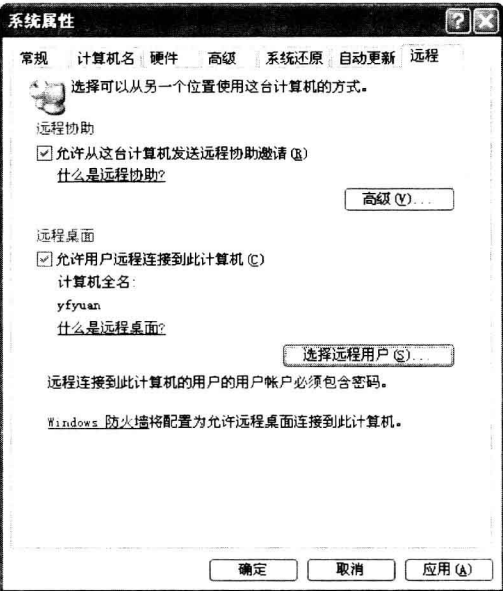


图 4-10 “远程”选项卡

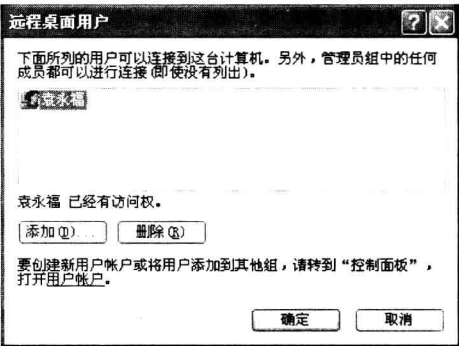


图 4-11 “远程桌面用户”对话框

在该对话框中，单击“添加”按钮会显示如图 4-12 所示的“选择用户”对话框。

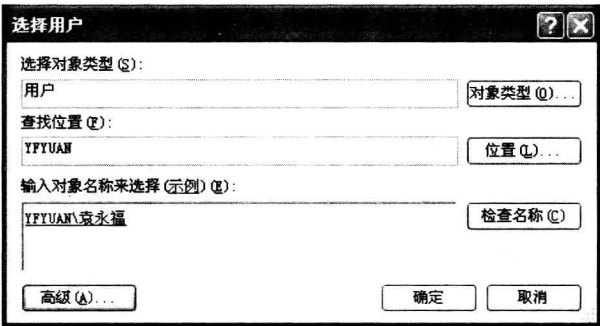


图 4-12 “选择用户”对话框

用户可以在该对话框中输入系统中已有的用户名，也可以单击“高级”按钮，再单击“立即查找”按钮来选择系统中已有的用户。

用户经过上述操作后，用户“袁永福”就能远程登录到这台计算机了。

上面的操作和用户界面都是在 Windows XP 中进行的，其他版本的 Windows 操作系统会有所不同，不过过程基本一样。

4.4.2 连接远程桌面

在客户端计算机中，运行程序“c:\windows\system32\mstsc.exe”，或者如图 4-13 所示，在

“运行”对话框中输入 `mstsc` 即可运行远程桌面客户端程序。

运行客户端程序后，首先显示的是如图 4-14 所示的登录对话框。

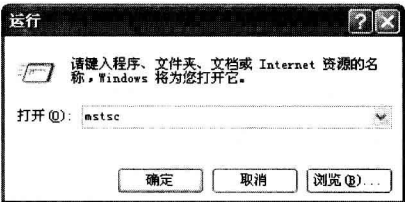


图 4-13 “运行”对话框

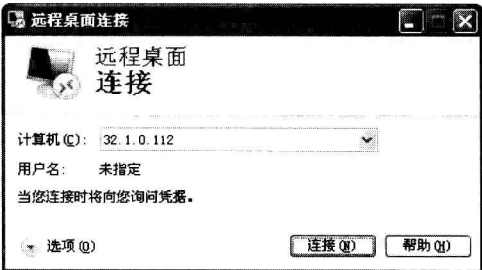


图 4-14 登录对话框

在该对话框中输入远程计算机的 IP 地址，单击“连接”按钮，程序便会连接远程计算机，此时连接界面如图 4-15 所示。

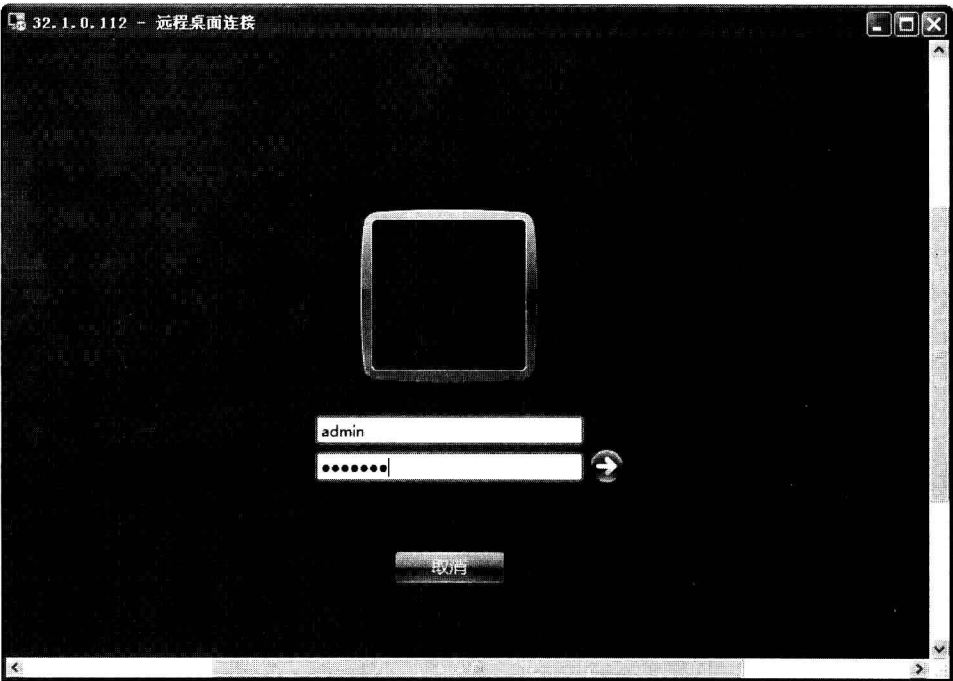


图 4-15 连接界面

此时用户需要输入管理员分配的用户名和密码进行远程桌面的登录，登录成功后，程序可进入远程计算机的桌面，如图 4-16 所示。

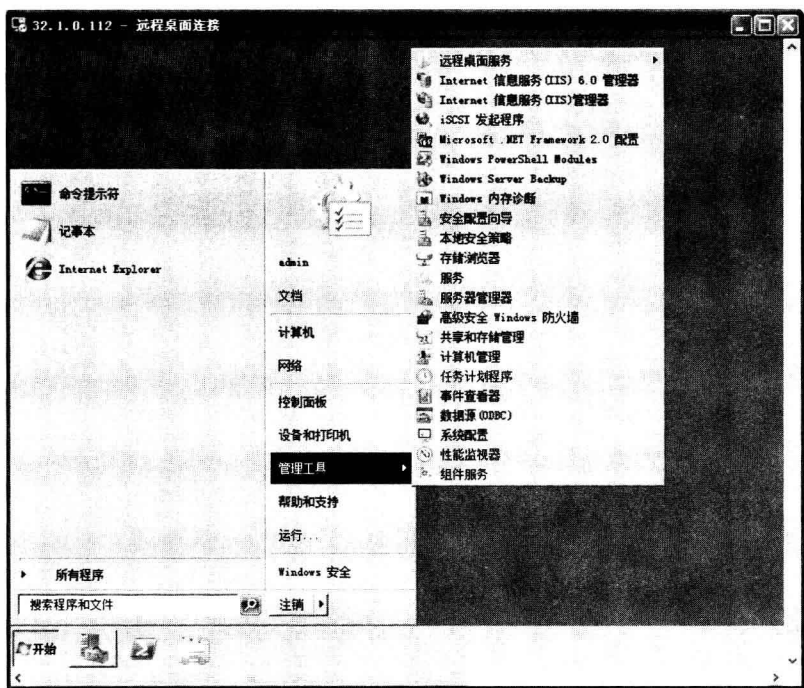


图 4-16 进入远程计算机的桌面

在该界面中，用户可以像操作本地计算机一样对远程的服务器进行任何操作。

操作完毕后，单击远程桌面客户端程序右上角的“关闭”按钮即可断开与远程桌面的连接，此时程序会显示如图 4-17 所示的提示对话框。

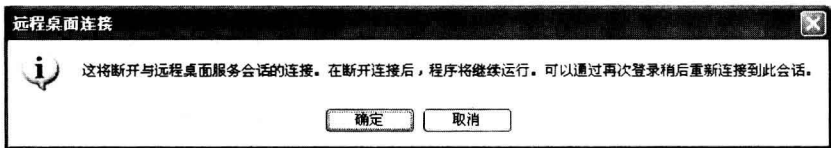


图 4-17 断开连接提示

单击“确定”按钮即可断开与远程桌面的连接。

注意，这里是断开与远程桌面的连接，而不是让远程桌面关闭。断开连接后，远程桌面还一直存在，其设置和用户界面保持不变，各种打开的窗体还在那里。当用户再次连接远程桌面时，还能看到上次连接时的用户界面。

4.5 任务管理器

Windows 任务管理器也是开发者经常用的程序，若开发者正在开发的应用程序还不成熟稳

定，那么程序死锁便是家常便饭，因此经常用任务管理器来杀死进程。此外开发者还经常用它来查看应用程序占有的内存量以及 CPU 的繁忙程度。

在 Windows 系统中，同时按“Alt+Ctrl+Delete”组合键即可运行任务管理器，其用户界面如图 4-18 所示。

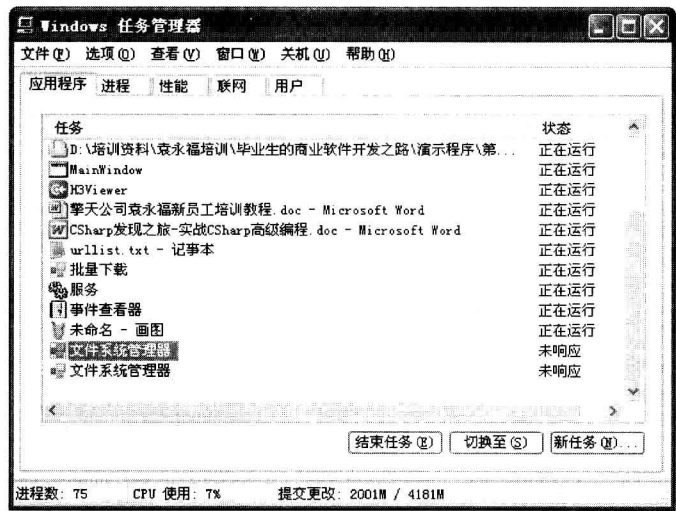


图 4-18 Windows 任务管理器

4.5.1 管理应用程序

任务管理器有多个选项卡，其中第一个是“应用程序”选项卡，能管理系统中正在运行的应用程序，该界面列出了系统中所有正在运行的应用程序的图标、名称和状态。处于死锁状态的应用程序就标记为“未响应”。未响应的程序多半正在执行艰难的操作或干脆“死”了。

用户单击“结束任务”按钮即可退出在列表中选择的应用程序，单击“切换至”按钮则能让这个应用程序的主窗体显示出来并获得输入焦点。

用户单击“新任务”按钮就会弹出如图 4-19 所示的“创建新任务”对话框。

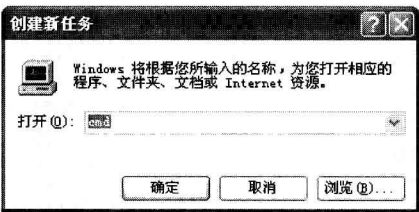


图 4-19 “创建新任务”对话框

用户可以在该对话框中输入命令行文本或选择一个可执行文件来运行新的程序。

4.5.3 查看系统性能

任务管理器的第三个选项卡是“性能”选项卡，其用户界面如图 4-23 所示。

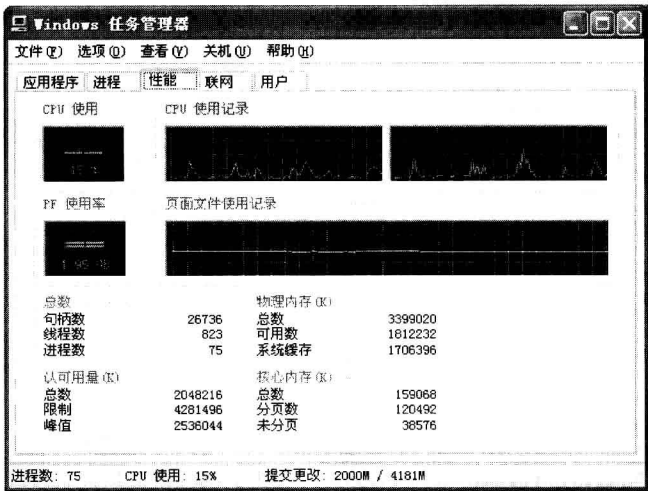


图 4-23 “性能”选项卡

在该界面中，用户可以查看 CPU 使用率和内存使用量的状态历史变化曲线图。若系统有多个 CPU，则会显示多个 CPU 的使用率曲线图，还列出了其他的系统参数，这个界面中的参数会每隔几秒钟刷新一次。

4.5.4 查看网络

任务管理器的第四个选项卡是“联网”选项卡，其用户界面如图 4-24 所示。



图 4-24 “联网”选项卡

该界面显示了系统中各个网络接口设备的使用量曲线图。

4.5.5 管理正在登录的用户

任务管理器的第五个选项卡是“用户”选项卡，其用户界面如图 4-25 所示。

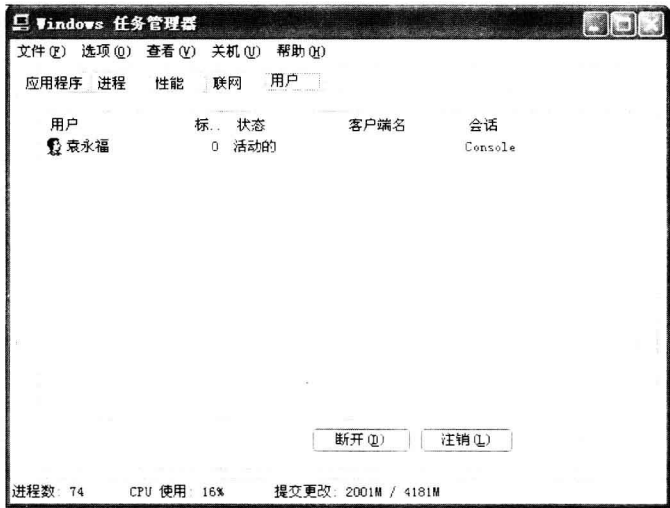


图 4-25 “用户”选项卡

在该界面中列出了系统中所有已经登录的用户信息，单击下面的“断开”按钮能断开用户连接，单击“注销”按钮能注销用户，关闭用户所使用的会话和桌面。

C#程序开发

本章介绍 C# 计算机编程语言的基础概念和常用的语法。

5.1 C#语言简介

C#（读 C Sharp）是微软主打的一种基于 .NET 平台的快速开发语言。它功能强大，使用方便，学习起来也比较快，已经成为一种主流的计算机软件编程语言了。

C# 是微软公司 2000 年 6 月发布的，是由 Anders Hejlsberg 领导研发的，此人是 Turbo Pascal 的主要作者，也是 Delphi 的创作者。

从语法上看，C# 是属于 C 语言家族的，与 Java 有不少共同点。现在有不少人认为 C# 抄袭 Java，不过现在多种现代编程语言之间是相互借鉴的，C# 包含了不少 VB 的特性，新版的 Java 也包含了不少 C# 特性。

从应用范围上看，有图有真相，图 5-1 显示了近年全球软件业界各种计算机编程语言的使用率变化情况。

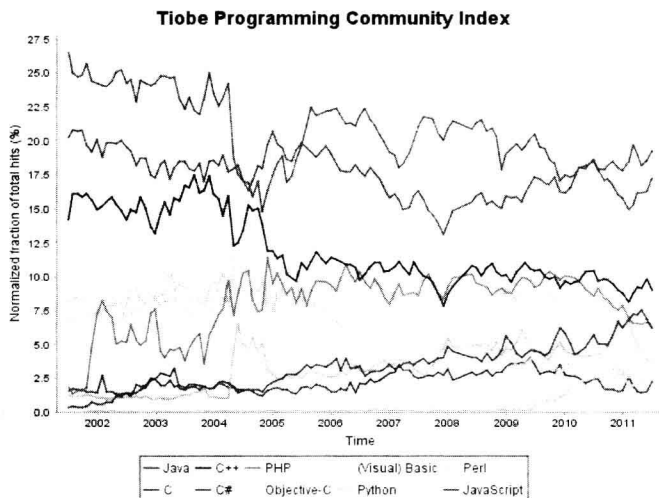


图 5-1 各类编辑语言使用率变化情况

在图 5-1 中，最黑的那条线是 C#的，可以看出，C#已经是一种主流的计算机编程语言，而且使用率还在稳步上升。表 5-1 是今年全球软件业界使用计算机编程语言的排行榜。

表 5-1 全球软件业界使用计算机编程语言的排行榜

Position May 2011	Position May 2010	Delta in Position	Programming Language	Ratings May 2011	Delta May 2010	Status
1	2	↑	Java	18.160%	+0.20%	A
2	1	↓	C	16.170%	-2.02%	A
3	3		C++	9.146%	-1.23%	A
4	6	↑↑	C#	7.539%	+2.76%	A
5	4	↓	PHP	6.508%	-2.57%	A
6	10	↑↑↑↑	Objective-C	5.010%	+2.65%	A
7	7		Python	4.583%	+0.49%	A
8	5	↓↓↓	(Visual) Basic	4.496%	-1.16%	A
9	8	↓	Perl	2.231%	-1.05%	A
10	11	↑	Ruby	1.421%	-0.67%	A

2001 年，ECMA 国际组织批准以微软的 C#编程语言作为标准，并于 2003 年被 ISO 国际组织确定为国际标准。类似的 C、C++语言都已经成为 ISO 国际标准，而 Java 尚未成为国际标准。

5.2 .NET 框架简介

.NET 框架是微软提出的先进的企业级应用开发框架，功能强大，使用方便，是 Windows 平台上最为流行的软件开发技术。

图 5-2 是微软.NET 框架结构图。

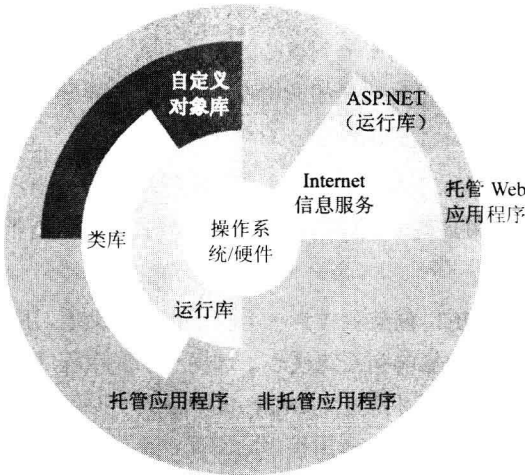


图 5-2 微软.NET 框架结构图

在该图中，Windows 操作系统/硬件形成整个架构的核心。其左边是运行库，类似 Java 虚拟机，在核心的基础上构筑了一个虚拟的计算机。.NET 程序是运行在这个虚拟机之上的，而不是直接运行在核心上。

运行库之上就是一个类库，这个类库包含了大量的功能类库，例如操作文件、数据库、网络、用户界面等。

类库是微软官方提供的，功能强大，但仍然不能满足千千万万的软件开发者的所有需求。此时开发者可以编写自定义对象库，实现各自的功能。左边最外层就是托管应用程序，它是用户实际使用的程序，它能调用类库和自定义对象库中的功能模块，实现各种完整的功能。

5.2.1 托管应用程序

托管应用程序的概念很重要，是 .NET 框架的基础概念之一。这个名词可以拆分成“托管”和“应用程序”。应用程序通俗地说就是 EXE 文件，可以通过运行来完成某种功能，关键是对“托管”这个概念的理解。

其实在现实中已经有托管这种概念，比如拿金融投资来说，某人手上有笔钱，他有两种投资方式，第一种就是偶然路过某个商场，发现商场的生意不错，于是出大手笔投资这个商场，然后撒手不管了，坐等年底分红；另一种就是把钱交给投资专业人士，比如股神巴菲特，然后撒手不管，坐等年底分红。

对于第一种投资方式，这是直接投资，那就是靠天吃饭了，完全依赖被投资的商场的运营，而投资者对此毫无了解，风险巨大，容易出大事。

对于第二种投资，这是间接投资，将钱交给专业人士，委托他管理这笔钱，也就是进行投资，这就是资金托管，虽然投资利益会被托管人分享，但风险小，不容易出大事。因此这种资金托管是比较好的。

类似地，应用程序也可以被托管。在应用程序没有被托管前，应用程序是直接运行在核心之上的，结果应用程序发生错误很容易影响到核心的和谐稳定。虽然企业应用是稳定压倒一切的，但没有一个开发者能开发出完全没有错误的应用程序，用户一直在冒着单个应用程序发生错误而影响整个系统的稳定，甚至整体死机的风险。

更坏的情况是，很多应用程序有安全漏洞，例如缓存区溢出漏洞，黑客利用应用程序的漏洞可以非法进入系统核心，给用户造成重要数据的泄露甚至丢失。这是非常严重的情况，是会引起纠纷的。

而应用程序被托管后，.NET 框架就成为一个软件错误防火墙，应用程序错误的波及范围会被 .NET 框架所限制，使其不致影响到系统核心，这样一个应用程序的崩溃就不会导致整个系统的死机。当应用程序有漏洞且被人利用后，.NET 框架会第一时间感知到应用程序的异常情况，对非法操作进行阻击，这样能避免托管应用程序的漏洞对系统核心造成的损害。

虽然应用程序被托管后性能受到一些影响，但能大幅提高系统核心的和谐稳定，因此在大部

分情况下是符合用户利益的。

5.2.2 微软中间语言规范

微软.NET 框架是微软推出的软件开发基础框架。它采用了类似 Java 的虚拟机技术，并进一步实现了中间语言运行技术，实现了微软中间语言规范（简称 MSIL），这让.NET 框架能支持多种计算机编程语言。其应用架构如图 5-3 所示。

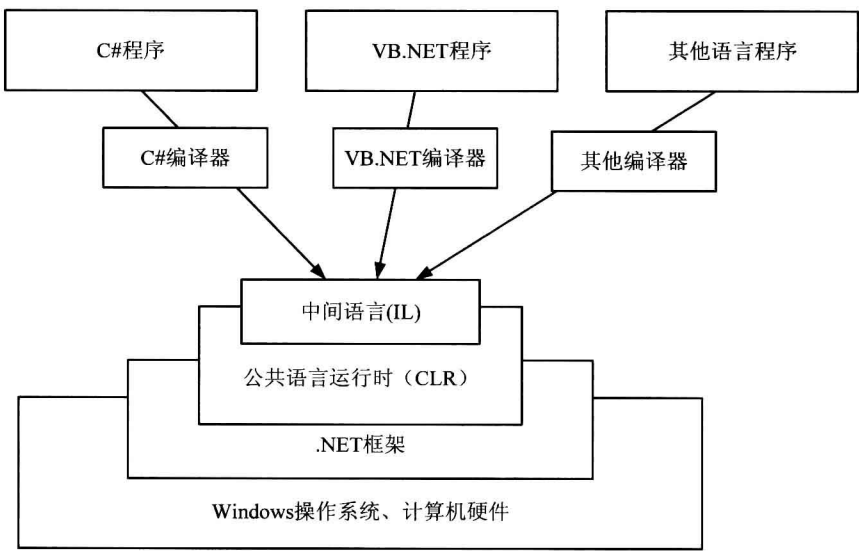


图 5-3 .NET 框架应用架构

对于 MSIL，初学者可以看成一种不限于特定硬件的抽象的标准汇编指令集，各种计算机编程语言编写的程序代码都可以编译成符合中间语言规范的程序。

目前 MSIL 已经被定为国际标准，任何人都可以实现符合这种标准的计算机编程语言及其配套编辑器。目前微软提供了 C#，VB.NET 和 J#的中间语言程序编译器，业界还存在其他编程语言的中间语言程序编译器。

微软针对中间语言实现了中间语言虚拟机，它能将 MSIL 指令转化为可直接在本机 CPU 上运行的机器指令，然后调用 CPU 执行这些指令。

为了在开发时和运行时提供很好的支持，微软在.NET 程序框架中添加了大量的实用软件组件库，还有用于实现 ASP.NET 的针对 IIS 的扩展程序库等重要软件模块。

Java 技术实现了跨操作系统平台但不跨语言，而.NET 技术实现了跨语言但不跨操作系统平台。理论上由于中间语言规范是国际标准，.NET 技术是可以跨平台的，而且业界著名的 MONO 项目也初步实现了.NET 技术的跨平台，但相信微软官方由于其商业利益是不会实现微软.NET 框架跨平台的。

综上所述，.NET 框架是一种软件开发框架，是一种虚拟的计算机，在其中能运行多种编程语言开发的应用程序；C#只是在.NET 框架上运行的其中的一种编程语言，但它是.NET 开发中最重要、最常用的编程语言。

使用 MSIL 技术对开发者来说会带来两大好处：

首先是在.NET 平台上进行开发不限制为某种具体的计算机编程语言。开发者可以使用除 C# 外的 VB.NET、Delphi.NET 等各种语言在其上进行开发，这样开发者此前掌握的非 C#语法知识和代码库就不会被废弃掉而能继续发挥作用，这包含过去的开发投资，利用已有的积累，很好地实现了开发语言技术的重用。

另外，MSIL 技术实现了一定的跨平台技术，基于.NET 平台开发的程序可以很方便地移植到各个版本的.NET 平台上，比如.NET CE、.NET CF 等，这样.NET 程序可以运行在 PC 服务器、PC、智能手机甚至单片机中，很好地实现了软件组件的重用。

第 6 章

C# 基本语法

本章是面向 C# 初学者的，不介绍 C# 的高级语法，只介绍 C# 的基础语法。要求读者已经掌握一种计算机高级编程语言，比如 VB 或 C++ 等，理解变量、数组、条件判断、循环等编程知识，达到国家计算机等级考试二级的水平。

6.1 C# 应用系统模块逻辑框架

C# 应用系统模块逻辑上分为应用系统、程序集、命名空间、类型、类型成员、功能语法块 6 层结构。

6.1.1 应用系统

应用系统是一个大的系统，它包含一个或多个程序集文件，还包含各种数据库、文档、资源等，例如系统设计文档、用户手册、培训资料。

6.1.2 程序集

程序集就是一个 C# 工程编译所得的 EXE 或 DLL 文件，它是 C# 程序功能模块单位。.NET 程序集保存在磁盘中就是一种二进制文件，符合 Windows 可执行文件格式。它包含以下三部分内容。

1. 元数据

.NET 程序集中的元数据用于进行程序集的自我描述，这些描述有 3 类：

- (1) 程序集本身的说明，包括名称、版本号、区域性、权限、所依赖的其他程序集信息。
- (2) 类型，本程序集中定义的所有类型及类型成员，以及这些对象的可见性等。
- (3) 特性，程序集中所有附加在程序集、类型、类型成员上的特性数据。

2. MSIL 指令数据

MSIL 指令就是 C# 编译后的程序代码，类似汇编指令，只能运行在 .NET 框架中。

当.NET 程序集运行时，.NET 框架会将 MSIL 代码进一步地编译成能直接运行在 CPU 硬件上的机器代码，然后让 CPU 执行代码。

3. 程序集资源数据

程序集中还可以包含一些嵌入的程序集资源文件，比如可以放置一张图像数据、字符串数据等。程序集资源数据可以看做程序集文件中的只读文件，程序集中的代码或其他代码都可以读取这些程序集资源文件中的数据，但不能修改数据。

6.1.3 命名空间

应用系统、程序集是所有.NET 程序的组织结构。命名空间、类型、类型成员和功能语法块就是 C#语法的内容了。

以下就是一个完整的 C#源代码内容。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("显示一个消息");
        }
    }
}
```

下面以这段代码为例说明 C#代码模块的组织结构。

命名空间是 C#最大的语法结构模块，这个模块是按照名称来标识的。每个类型都有所属的命名空间。同一个命名空间中的类型不能重名，但不同的命名空间可以分别包含相同名称的类型。

在 C#中以下面的语法结构来使用命名空间。

```
namespace 命名空间名称
{
    若干个类型
}
```

在本例中，使用代码 “namespace WindowsFormsApplication1 ” 定义了一个名为

“WindowsFormsApplication1”的命名空间，其中定义了名为“Form1”的类型。

对于很多编程语言，比如 C 语言，它没有命名空间的概念，几千个函数堆积在一起；VB 也是几百个类混在一起的，这些对开发人员记忆和调用都非常不方便。

而使用命名空间，能将诸多类型层次化，有利于模块化，也方便调用。

命名空间可以分层次，比如以下代码：

```
namespace 命名空间名称
{
    若干个类型
}
namespace 命名空间名称.子命名空间
{
    若干个类型
}
```

这段代码定义了一个命名空间和子命名空间，对层数是没有限制的。

在 C#功能代码中，要引用某个类型，必须用它的类型全名，也就是“命名空间.类型名称”，这样代码显得比较臃肿，而且若未来类型的命名空间发生改变，则需要修改不少代码。此时 C#使用 using 关键字来引用命名空间。

例如在演示代码的开头有一行代码“using System.Windows.Forms;”，此时就在代码中引用了命名空间“System.Windows.Forms”，而窗体类型“Form”就在这个命名空间下面，可以在代码中直接使用“Form”来获得窗体的类型。若没有这个 using 代码，则必须使用“System.Windows.Forms.Form”来获得窗体类型，这样代码就显得有些臃肿。在这里，using 关键字的作用有点类似 VB 中的 with 关键字。

注意，using 关键字除了能引用命名空间外，还能形成自动销毁对象的语法结构。对于这种写法将在后面描述。

6.1.4 类型

命名空间下面就是定义类型，包括类类型、结构体类型、枚举类型、委托类型、接口类型和范型。命名空间下面不能直接跟上字段、方法等。因此，C#没有游离于类型之外的方法。

理论上可以跳出命名空间而直接编写类型，这样也能编译通过，但在实践中不推荐这么做。

在演示代码中，使用以下代码定义了一个类型。

```
public partial class Form1 : Form
{
}
}
```

在这段代码中：

- public 关键字说明该类型是公开的，可以被任何程序调用。
- partial 关键字说明这段代码不是这个类型的全部代码，C#工程中还有其他代码文件包含了这个类型的代码。在编译程序的时候，编辑器会将这些分散的源代码收集起来组

成一份完整的源代码参与编译。

- `class` 关键字说明这是在定义一个类类型。
- `Form1` 是新类型的名称。
- `Form` 跟在 `Form1` 后面，中间有一个冒号，说明新增的类型 `Form1` 继承已有的类型 `Form`。若前面没有代码 “`using System.Windows.Forms.Form`” 引用命名空间，则该行代码必须写成 “`public partial class Form1 : System.Windows.Forms.Form`”。
- `Form` 后面跟着一对花括号，定义了它的类型成员及区域。

6.1.5 类型成员

类型下面就是定义类型的成员了，包括字段、属性、索引器、方法和事件。

在演示代码中，使用以下代码定义了一个名为 `Form1` 的成员方法。

```
public Form1()
{
    InitializeComponent();
}
```

该方法名称和类型名称一样，而且没有定义返回值，说明该方法就是该类型的构造函数。

以下代码定义了一个成员方法。

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("显示一个消息");
}
```

在这段代码中：

- `private` 关键字说明该方法是私有的，只能在本类型内部使用，类型之外不能使用。
- `void` 关键字说明该方法没有任何返回值。
- `button1_Click` 是该方法的名称。
- “`object sender`” 定义了该方法的第一个参数，`object` 是参数类型，`sender` 是参数名；类似地，“`EventArgs e`” 定义了第二个参数，两个参数之间用逗号分开。

函数声明后面就是函数体了。在这里使用了一行代码 “`MessageBox.Show("显示一个消息");`”，这行代码调用了类型 `MessageBox` 的 `Show` 方法，参数是字符串“显示一个消息”。这行代码的作用就是调用类型 `System.Windows.Forms.MessageBox` 的 `Show` 静态方法来显示一个文本消息框。

注意，在写 C# 代码时，根据需要在分号 “`;`”。

6.1.6 功能语法块

类型成员的属性和方法内部就是功能代码模块了。它是由若干个表达式、循环结构、判断结构、异常处理结构等多种语法结构组成的。后续章节有其详细说明。

6.2 数据类型

任何编程语言都有数据类型的概念，这些数据类型大体可分为字符串、文本、数字、日期等。图 6-1 是 C#中数据类型的继承关系图。

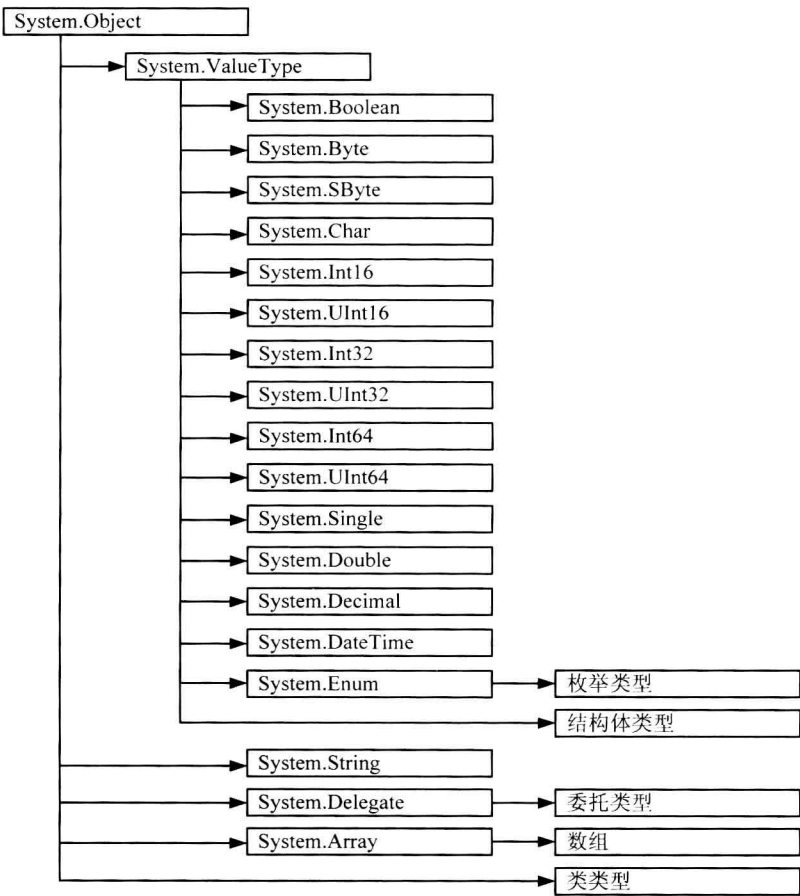


图 6-1 C#中数据类型的继承关系图

在这个结构图中所有以“System”开头的都属于基础数据类型，其他的都是自定义数据类型。

6.2.1 基础数据类型

System.Object 类型是 C#数据类型体系中最为基础的类型，在 C#中使用关键字“object”表示该类型。

C#的数据类型体系和其他编程语言的最大不同就是实现了各种数据类型的统一，并提供了一个所有数据类型的基础类型，那就是在命名空间 System 下面的 Object 类型。

在其他编程语言中，例如 VB，其整数、浮点数等基础类型都是原子类型，不是从其他类型派生的。但 C#彻彻底底地实现了面向对象编程，即使这些基础类型都是从 Object 类型派生过来的，Object 类型就是最基础的类型，再往上就没有类型了。

Object 类型提供一些成员方法，最常见的如表 6-1 所示。

表 6-1 Object 类型提供的成员方法

成员方法	说明
Equals	带一个参数，用于对两个对象数据进行比较，若相等则返回 True，否则返回 False
Finalize	在自动回收对象之前执行清理操作，该方法一般由 .NET 框架自动调用
GetHashCode	生成一个与对象的值相对应的数字以支持哈希表的使用
ToString	生成描述对象数据的字符串，一般供人阅读

在 Object 类型的基础上 C#定义了如表 6-2 所示的基础数据类型，而且有很多基础数据类型在 C#中用对应的关键字表示。

表 6-2 基础数据类型

类 型	对应的 C#关键字	说 明
System.Boolean	bool	布尔类型，其值只能为 true 或 false，该数据类型占用 1 字节内存
System.Byte	byte	表示一个取值范围从 0 到 255 的整数，该数据类型占用 1 字节内存
System.SByte	sbyte	表示一个从-128 到 127 的整数，占用 1 字节
System.Char	char	表示一个字符数据，占用 2 字节。与 C 语言类似，Char 类型可以强制转换为整数。这个字符数据采用 Unicode 编码格式
System.Int16	short	表示一个从-32 768 到 +32 767 的整数，占用 2 字节
System.UInt16	ushort	表示一个从 0 到 65 535 的整数，占用 2 字节
System.Int32	int	表示一个从-2 147 483 648（约负 21 亿）到 +2 147 483 647（约 21 亿）的整数，占用 4 字节
System.UInt32	uint	表示一个从 0 到 4 294 967 295 的整数，占用 4 字节
System.Int64	long	表示一个从-9 223 372 036 854 775 808 到 +9 223 372 036 854 775 807 的整数，占用 8 字节
System.UInt64	ulong	表示一个从 0 到 18 446 744 073 709 551 615 的整数，占用 8 字节
System.Single	float	表示一个从-3.402823e38 到+3.402823e38 的单精度浮点数字，有 7 位有效数字，占用 4 字节
System.Double	double	表示一个从-1.79769313486232e308 到 +1.79769313486232e308 的浮点数，有 15 位有效数字，占用 8 字节
System.Decimal	decimal	表示一个从+79 228 162 514 264 337 593 543 950 335 到-79 228 162 514 264 337 593 543 950 335 的数字，而且计算时尽量不进行舍入操作，这样能维护运算精度，比较适合财务运算

续表

类 型	对应的 C#关键字	说 明
System.DateTime	无	表示一个从公元（基督纪元）0001 年 1 月 1 日午夜 12:00:00 到公元（C.E.）9999 年 12 月 31 日晚上 11:59:59 的时间日期数据，精确到 100 纳秒 在初始化日期数据的时候，可以传递年数、月份数和日数，也可以继续添加 24 小时制的小时数、分钟数和秒数 以下代码定义了一个 DateTime 类型： DateTime dtm = new DateTime(1980 , 2 , 14); 也可以为 DateTime dtm2 = new DateTime(1980 , 2 , 14 , 16.23 , 39);
System.String	string	表示一段文本，采用 UTF-16 编码，可以包含字符 “\0”
System.Enum	enum	所有枚举类型的基础类型
System.Delegate	delegate	所有委托类型的基础类型
System.Array	无	所有数组类型的基础类型

6.2.2 数组

C#支持数组，它包含若干个相同类型的变量。在 C#代码中，使用 “type[] arrayName” 的方式来定义数组。例如以下代码定义了几个数组。

```
int[ ] ids = null; //定义了一个整数数组
string[ ] names = new string[ 100 ]; //定义了一个字符串数组，并初始化为包含 100 个数据
```

在初始化数组时，数组里的元素值也被初始化了。对于 bool 类型的数组，其元素值初始化为 false，数值型的初始化为 0，字符串的初始化为 null。

在 C#中数组的下标是从 0 开始的，可以使用数组对象的 Length 属性获得数组的长度。例如以下代码就是遍历数组中所有元素的。

```
string[ ] names = new string[100];
for (int iCount = 0; iCount < names.Length; iCount++)
{
    string name = names[iCount];
}
```

也可以使用 foreach 语句来遍历数组中的所有元素。如下所示：

```
string[ ] names = new string[100];
foreach( string name in names )
{
    //此处可以使用变量 “name” 的值
}
```

注意，在 for 循环中可以通过修改 “names[iCount]” 的值来修改数组中存储的数据。但在 foreach 循环中，name 不能修改，是只读的。

在初始化结构体类型的数组时，数组是有效的，而且数组中的元素也是有效的；而在初始化类类型的数组时，虽然数组是有效的，但其数组的元素全部为空引用。

例如以下代码定义了结构体类型 `MyPeopleStruct`。

```
public struct MyPeopleStruct
{
    public string Code;
    public string Name;
}
```

针对该类型执行以下代码是不会出事的。

```
MyPeopleStruct[] peoples = new MyPeopleStruct[100];
peoples[33].Name = "张三";
```

因为结构体数组初始化后，系统会自动创建数组元素对象。

例如以下代码定义了类类型 `MyPeopleClass`。

```
public class MyPeopleClass
{
    public string Code = null;
    public string Name = null;
}
```

若有以下代码：

```
MyPeopleClass[] peoples = new MyPeopleClass[100];
peoples[3].Name = "张三";
```

这段代码可以编译通过，但运行时会在代码 “`peoples[3].Name = "张三";`” 处报空引用的程序错误。这是因为初始化类类型数组，只是为数组分配了内存空间，并没有相应地创建对象类型，该数组的元素全部设置为空引用，对于空引用调用其成员就会报空引用错误。

空引用错误是 C# 程序中最常见的错误。

6.2.3 自定义类型

在程序开发中，仅仅使用这些基础数据类型是不够的，开发人员还要根据需要自定义数据类型。在 C# 中，自定义的数据类型有类、结构体、枚举、接口、委托和范型。

1. 类类型

在 C# 中使用关键字 “`class`” 定义的类型为类类型，以下代码就定义了一个类类型。

```
public class PeopleClass
{
    public PeopleClass()
    {
    }

    private string _Name = null;
    public string Name
    {
        get
        {
            return _Name;
        }
        set
        {
        }
    }
}
```

```

        _Name = value;
    }
}
public override string ToString()
{
    return _Name;
}
}

```

在代码“`public class PeopleClass`”中，关键字“`public`”说明该类型是公开的；“`class`”说明这是一个类类型；“`PeopleClass`”指定了类型的名称。

类类型是引用类型，比如执行了以下代码：

```

PeopleClass p1 = new PeopleClass();
p1.Name = "张三";
People p2 = p1 ;
p2.Name = "李四";

```

则变量 `p1` 和 `p2` 指向的是同一个对象，也可以理解为指向同一个内存地址。显然，通过变量 `p2` 修改值和通过 `p1` 修改值其效果是一样的，这样 `p1.Name` 和 `p2.Name` 的值都等于“李四”。

2. 结构体类型

C#支持结构体类型，以下代码就定义了一个结构体类型。

```

public struct PeopleStruct
{
    public string Code;
    public string Name;
    public bool Sex;
}

```

这段代码定义了一个名为 `People` 的结构体，它有“`Code`”、“`Name`”、“`Sex`”三个公开字段。我们可以使用以下代码来使用这个结构体类型。

```

PeopleStruct myPeople = new PeopleStruct( );
myPeople.Code = "1000";
myPeople.Name = "张三";

```

结构体类型中可以定义字段、属性、方法和事件，但和类类型有以下区别：

- 结构体中字段不能初始化，比如在结构体中需要写成“`public string Code;`”，而在类类型中可以写成“`public string Code = null;`”。
- 结构体中不能定义默认的构造函数（无参数的构造函数）。
- 结构体可以定义带参数的构造函数，但在函数体中执行任何代码前必须对所有字段进行赋值初始化。
- 结构体对象在赋值时会重新创建一个对象并复制所有字段值。对新结构体数据的修改不会影响原始对象的内容。如果对结构体类型 `People` 执行以下代码：

```

People p1 = new People();
p1.Name = "张三";
People p2 = p1 ;
p1.Name = "李四";

```

则 p2.Name 的值还为“张三”。

- 结构体不能继承自其他结构体类型，也不能派生新的结构类型。所有结构体类型直接继承自类型 System.ValueType。

3. 枚举类型

C#支持枚举类型，以下代码就定义了一个枚举类型。

```
public enum BarcodeStyle
{
    Code128A,
    Code128B,
    Code128C
}
```

在 C#中，枚举变量是可以转换为整数的，第一个枚举成员默认等于 0，然后依次增加，不过也可以明确地设置枚举成员数值。例如上述代码等价于：

```
public enum BarcodeStyle
{
    Code128A = 0,
    Code128B = 1,
    Code128C = 2
}
```

设置数值是比较自由的，可以任意指定，例如可以使用以下代码定义枚举类型。

```
public enum FlagStyle
{
    Flag1 = 1,
    Flag2 = 4,
    Flag3 = 64
}
```

所有枚举类型都是从基础类型 System.Enum 派生的。System.Enum 类型提供了一些能用于所有枚举类型的方法，常用的如表 6-3 所示。

表 6-3 常用枚举类型

枚举类型	说 明
GetName	获得指定数据的枚举项目的名称。该函数是静态的，具有两个参数：第一个是枚举类型，第二个是某个常数 例如对于上面的 BarcodeStyle 枚举类型，执行代码“Enum.GetName(typeof(BarcodeStyle), 0)”，就返回字符串“Code128A”；执行“Enum.GetName(typeof(BarcodeStyle), 1)”就返回字符串“Code128B”
GetNames	获得由枚举类型的所有枚举项目名称组成的字符串数组。该函数是静态的，参数是枚举类型变量 例如对于 BarcodeStyle 枚举类型，执行代码“Enum.GetNames(typeof(BarcodeStyle))”就返回一个字符串数组，数组元素是“Code128A”、“Code128B”、“Code128C”
GetValues	获得由枚举类型的所有枚举项目组成的数组。该函数是静态的，参数是枚举类型变量 例如对于 BarcodeStyle 类型，执行代码“Enum.GetValues(typeof(BarcodeStyle))”就返回一个数组，数组元素就是“BarcodeStyle.Code128A”、“BarcodeStyle.Code128B”、“BarcodeStyle.Code128C”

续表

枚举类型	说 明
Parse	<p>解析字符串并转化为枚举类型，若转化失败则抛出异常。该函数是静态的，参数是指定的枚举类型和要解析的字符串，此外还有第三个布尔类型的可选参数，用于指明是否区分大小写</p> <p>例如对于 BarcodeStyle 类型，执行代码“Enum.Parse(typeof(BarcodeStyle), “Code128A”)”就返回“BarcodeStyle.Code128A”。执行代码“Enum.Parse(typeof(BarcodeStyle), “code128a” , true)”也返回“BarcodeStyle.Code128A”</p> <p>注意，当解析失败时，该函数会抛出异常</p>
TryParse	<p>解析字符串并试图将其转化为枚举类型，若转化失败则不抛出异常，该函数返回转化是否成功的布尔值。该函数是静态的：第一个参数就是要转化的字符串，第二个可选参数用于指明转化时是否区分大小写，第三个参数就是保存转化结果的枚举变量</p> <p>例如对于“BarcodeStyle”，调用代码“Enum.TryParse(“Code128A” , out value)”，则函数返回 true，而且 value 值被设置成“BarcodeStyle.Code128A”；调用代码“Enum.TryParse(“abc” , out value)”，则函数返回 false，表明转化失败</p>
ToString	返回表示枚举值的字符串，一般为其所表示的枚举项目的名称

4. 接口类型

在 C#中使用关键字“interface”定义的数据类型为接口类型，以下代码就定义了一个接口类型。

```
public interface IMyInterface
{
    string Value
    {
        get;
        set;
    }
    int Sum( int a , int b );
}
```

在这段代码中，“public”说明该类型是公开的；“interface”说明该类型是接口类型；“IMyInterface”指定类型名称，一般约定成俗，接口类型的名称是以大写字母“I”开头的；“string Value { get ; set ;}”定义了一个属性；“int Sum(int a , int b)”定义了一个方法。

接口用于定义一组属性、方法和事件，但不能包含字段。接口的成员无须修饰，全部是公开的。

接口不能实例化，仅仅是一个空洞的模板让人敬仰，但毫无作用；它必须“深入群众”，“灵魂”附在其他类或结构体中才能发挥作用。这个灵魂附体的过程就是实现接口。

实现接口的方式不复杂，就是在定义类型的时候声明是从接口类型派生的，并将接口中定义的所有属性、方法一个不落地实现，从这个角度看，接口类型有点像类类型。

例如对于上面的 IMyInterface 类型，使用以下代码就定义了一个实现该接口的类。

```
public class MyClass2 : IMyInterface
{
    private string _Value = null ;
```

```
public string Value
{
    get
    {
        return _Value ;
    }
    set
    {
        _Value = value ;
    }
}

public int Sum(int a, int b)
{
    return a + b ;
}
}
```

当接口成员比较多，难以记忆时，VS.NET 的 C#代码编辑器会提供一些帮助功能，来编写实现接口的代码。如图 6-2 所示，文本光标移动到方框所在的“IMyInterface”，此时其左下角出现一个智能标签，用鼠标单击这个智能标签会显示一个菜单。

```
public interface IMyInterface
{
    string Value
    {
        get;
        set;
    }
    int Sum( int a , int b );
}

public class MyClass2 : IMyInterface
{
    //
}

实现接口 “IMyInterface” (I)
显式实现接口 “IMyInterface” (X)
```

图 6-2 智能标签帮助编写实现接口的代码

单击“实现接口 ‘IMyInterface’”菜单项目则自动地往“MyClass2”的代码块中插入能实现IMyInterface的代码，此时自动生成的代码如下：

```
public class MyClass2 : IMyInterface
{
    #region IMyInterface 成员

    public string Value
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }

    public int Sum(int a, int b)
```

```

        {
            throw new NotImplementedException();
        }

        #endregion
    }

```

若单击“显式实现接口 ‘IMyInterface’”菜单项目则自动生成以下代码：

```

public class MyClass2 : IMyInterface
{
    #region IMyInterface 成员

    string IMyInterface.Value
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }

    int IMyInterface.Sum(int a, int b)
    {
        throw new NotImplementedException();
    }

    #endregion
}

```

实现接口和显式实现接口是有区别的。对于实现接口，可以通过以下方式访问它的成员。

```

MyClass2 instance = new MyClass2( );
instance.Sum(3, 4);

```

也可以通过以下方式访问成员。

```

IMyInterface instance = new MyClass2();
instance.Sum(3, 4);

```

或者

```

MyClass2 instance = new MyClass2( );
IMyInterface instance2 = (IMyInterface )instance;
instance2.Sum(3, 4);

```

而对于显式实现接口只能使用第二种方式访问。

在实际开发中，这两种实现接口的方式还可以混用，不过显式实现接口用得比较少。

在 C#中，类只能继承另外一个类，但可以实现多个接口。与此不同的是，C++可以继承自多个类和多个接口。

5. 委托类型

委托就是一个指向成员方法的对象，C 语言中有一个函数指针的概念，委托就可以看做面向

对象的函数指针。其语法为“**delegate** 返回值 委托名(参数类别)”。例如以下代码就定义了一个委托类型。

```
public delegate int Int2Handler( int a , int b );
```

在这段代码中，“**public**”指明该类型是公开的，“**delegate**”说明正在定义一个委托类型，“**int**”指明委托的返回类型，“**Int2Handler**”是委托类型的名称，“**int a , int b**”是该委托使用的参数列表。

委托代表一个类型成员方法，而且该方法的返回值和参数列表必须和委托声明的一模一样。

例如在某个类中定义了以下三个成员方法：

```
private int Sum(int a, int b)
{
    return a + b;
}

private int Mul(int a, int b)
{
    return a * b;
}

private double Sum2(double a, double b)
{
    return a + b;
}
```

则方法 **Sum** 和 **Mul** 的签名与委托 **Int2Handler** 的签名完全吻合，因此 **Int2Handler** 委托可以指向这两个方法；而方法 **Sum2** 的签名不对，因此 **Int2Handler** 委托不能指向该方法。

以下代码演示了使用委托：

```
Int2Handler handler = null;
handler = new Int2Handler(Sum);
int result = handler(3, 4);           //返回 7
handler = new Int2Handler(Mul);
result = handler(3, 4);               //返回 12
```

在这段代码中，创建委托对象实例传入的参数就是方法名称，因此同样执行代码“**handler(3 , 4)**”。由于第一次委托指向方法 **Sum**，计算结果为 7；而第二次委托执行方法 **Mul**，计算结果为 12，这样委托就能实现非常好的灵活性。

C#中可以实现匿名委托，以下代码就演示了匿名委托。

```
Int2Handler handler = delegate(int a, int b)
{
    return a + b;
};
int result = handler(3, 4);

handler = delegate(int a, int b)
{
    return a * b;
};
result = handler(3, 4);
```

可以看出使用“委托类型 委托变量名 = delegate(参数列表) { 方法功能代码 };”语句可以创建一个匿名委托，而且无须编写独立的方法来放置功能代码。因此使用匿名委托能写出比较精巧的代码，但要注意匿名委托也有缺点，那就是不利于调试修改。

VS.NET 提供断点调试时修改程序代码的功能，VB 也有类似的功能。但当修改了包含匿名委托方法中的代码时，VS.NET 的这个功能就无效了，需要重新编译、重新运行程序，才能应用代码的改变。

6. 泛型

泛型是一种比较新的软件技术，对于开发者来说，就是一套类型代码模板，在开发的时候使用某种尚未确定的数据类型，就用一种虚拟的类型来替代；在运行的时候可以将其操作的类型替换成指定的具体类型。

在开发中最常用的泛型类型为“System.Collections.Generic.List<>”和“System.Collections.Generic.Dictionary<>”。泛型类型在实例化时必须指明其采用的数据类型，而且实例化后的数据类型是特定的，不能修改。

例如对于类型“System.Collections.ArrayList”和“System.Collections.Generic.List<>”功能类似，下面进行对比。

以下代码使用了 ArrayList 类型。

```
//没有使用泛型，可以添加任意类型的元素
System.Collections.ArrayList list1
    = new System.Collections.ArrayList();
list1.Add(new MyPeopleClass());
list1.Add(new MyPeopleStruct());
//列表元素必须强制类型转换
((MyPeopleClass)list1[0]).Name = "张三" ;
```

在这段代码中，开发者可以在 ArrayList 列表中添加任何类型的对象，而且调用列表中元素的属性时需要进行强制类型转换。这个过程比较灵活，但很不安全，很容易导致类型转换出错和运行时出错。

以下代码使用了泛型列表“List<>”：

```
//使用泛型
System.Collections.Generic.List<MyPeopleClass> list2
    = new System.Collections.Generic.List<MyPeopleClass>( );
list2.Add(new MyPeopleClass());
//list2.Add(new MyPeopleStruct()); 这样写有编译错误
list2[0].Name = "张三"; //无须强制类型转换就能访问元素成员
```

在这段代码中，代码“new System.Collections.Generic.List<MyPeopleClass>()”创建了一个列表对象，其中“<MyPeopleClass>”指明了列表元素数据类型固定为“MyPeopleClass”类型。接着代码调用列表对象的 Add 方法向列表中添加成员，由于在初始化时已经限制列表元素为 MyPeopleClass 类型，因此 Add 方法只能接受 MyPeopleClass 类型的参数，若使用其他数据类型的参数则会报编译错误。

由于已经限制了列表元素成员的数据类型，因此无须进行强制数据类型转换就能调用列表元素的方法，代码“list2[0].Name = "张三"”就展示了这种技术。

使用泛型能将很多运行时错误转换为编译时错误，这可以将很多程序错误消灭在萌芽中，提高程序的开发速度，同时还能提高程序运行性能，因此在开发中应尽量采用泛型。

6.3 数据类型转换

在程序开发中经常需要进行数据类型的转换，C# 提供强制数据类型转换和使用关键字 `as` 的数据类型转换，还提供使用关键字 `is` 来进行数据类型的判断。

6.3.1 强制类型转换

强制类型转换就是将一个类型的数据无条件地强制转换为其他数据类型。有安全的强制类型转换和不安全的强制类型转换两种。

对于数值类型的基础数据类型，存在以下安全的强制数据类型转换路径“`bool`→`sbyte`→`char`→`short`→`int`→`long`→`float`→`double`”，从这里可以看出，这条安全的路径越靠后，取值范围就越大。例如以下代码就是一个安全的强制类型转换。

```
int v1 = 98;
double v2 = (double)v1;
```

对于安全的强制类型转换，可以将类型转换运算符省略，比如上述第二行代码可以写成“`double v2 = v1;`”。

反过来，在这个强制类型转换路径上逆道行驶，就是不安全的强制类型转换。例如以下代码就是不安全的强制类型转换。

```
double v3 = 98.34;
int v4 = (int)v3;
```

转换后，数值“98.34”被转换为“98”，两者不等，因此是不安全的，可能导致程序出错，而且不安全的强制转换中转换符号不能省略，比如上面的第二行代码就不能写成“`int v4 = v3;`”。

类类型之间也能进行强制类型转换，语法为“类型名称 变量名=(类型名称)变量名 2”，而且要保证转换对象引用的是转换后的类型或派生类。

若在程序中定义了以下几个类型，则各个类型的继承派生关系如图 6-3 所示。

这几个类型按照继承关系形成一个以“`System.Object`”类型为唯一根节点的树状结构，从叶子节点出发到跟节点，形成多条安全的数据类型转换路径，例如“卡车类型→汽车类型→机械类型→`Object`”，以及“大米类型→食品类型→`Object`”，沿着这条路径进行类型转换是安全的，肯定能转换成功；反过来是不安全的类型转换，可能不成功。

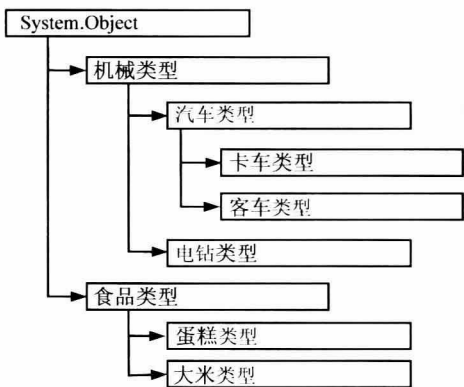


图 6-3 各个类型的继承派生关系

例如以下转换过程是安全的，肯定成功。

```
卡车类型 instance = new 卡车类型( ) ;
机械类型 instance2 = ( 机械类型 ) instance ;
```

以下转换过程是不安全的，可能会失败。

```
机械类型 instance = new 卡车类型( ) ;
机械类型 instance2 = new 电钻类型( ) ;

汽车类型 car1 = ( 汽车类型 ) instance1 ; //不安全的转换，能成功
汽车类型 car2 = ( 汽车类型 ) instance2 ; //不安全的转换，不成功，会报运行时错误
```

若在这个类型继承树上凭空跳到另外一个节点，那么就是不可能的类型转换，而且会报编译错误。例如以下的类型转换的代码就根本行不通。

```
卡车类型 instance = new 卡车类型( ) ;
食品类型 instance2 = ( 食品类型 ) instance ;
```

6.3.2 as 类型转换

由于强制类型转换可能会发生错误，故 C#提供了一个使用关键字 as 的类型转换。其语法为“要转换的类型名称 变量= 要转换的变量 as 要转换的类型名称”。若转换成功则会设置变量值，若转换失败则设置变量值为空引用。

例如对于上面提到的数据类型转换路径“卡车类型→汽车类型→机械类型→Object”，则可以执行以下的代码：

```
汽车类型 instance = new 卡车类型( ) ;
客车类型 bus = instance as 客车类型 ;
```

由于这是不安全的转换，转换失败，此时程序不会报错，但会设置变量 bus 为空引用。

以下代码：

```
汽车类型 instance = new 卡车类型( ) ;
卡车类型 instance2 = instance as 卡车类型；
```

是安全的转换，转换成功，会设置变量 `instance2` 指向某个对象实例。

as 类型转换比强制类型转换的好处就是转换失败时不会发生错误，此时应用程序需要检查转换结果值来判断转换是否成功。

6.3.3 is 类型判断

C# 提供关键字 `is` 来进行类型判断，其语法为“变量名 `is` 类型名称”，这是一个表达式，返回的是布尔值，用于判断指定的对象实例可否转换为指定的类型。例如执行以下代码：

```
机械类型 instance = new 卡车类型();  
bool result = instance is 汽车类型;
```

由于此处 `instance` 指向的是一个卡车类型，而卡车类型是汽车类型的派生类，可以转换为汽车类型，因此 `instance` 类型是汽车类型，而 `result` 的值为 `true`。

使用关键字 `is` 进行类型判断，能判断对象是否是指定的类型或派生类型，也可以判断是否实现了指定的接口，而且这个过程不会报错。

6.4 可访问级别

在说明自定义类型时提到了类型及成员的可访问级别，下面进行详细说明。以下是关于可访问级别的演示代码，这段代码保存在本章配套程序的代码文件“可访问级别.cs”中。

```
public class 可访问级别范例  
{  
    /// <summary>  
    /// 私有成员  
    /// </summary>  
    private string _PrivateField = null;  
  
    /// <summary>  
    /// 受保护的成员  
    /// </summary>  
    protected string _ProtectedField = null;  
  
    /// <summary>  
    /// 公开的成员  
    /// </summary>  
    public string _PublicField = null;  
  
    /// <summary>  
    /// 程序集内部成员  
    /// </summary>  
    internal string _InternalField = null;  
  
    public void Test()  
    {  
        this._PrivateField = "能访问 private 成员";  
        this._ProtectedField = "能访问 protected 成员";  
        this._InternalField = "能访问 internal 成员";  
    }  
}
```



```

        this._PublicField = "能访问 public 成员";
    }
}

public class 测试可访问级别范例
{
    public void Test()
    {
        可访问级别范例 instance = new 可访问级别范例();
        instance._InternalField = "能访问 internal 成员";
        instance._PublicField = "能访问 public 成员";
        // 不能访问 _PrivateField , _InternalField , _ProtectedField
    }
}

public class 测试可访问级别范例2 : 可访问级别范例
{
    public void Test2()
    {
        this._InternalField = "能访问 internal 成员";
        this._ProtectedField = "能访问 protected 成员";
        this._PublicField = "能访问 public 成员";
        //不能访问 this._PrivateField
        //如果是在另外一个C#项目中定义了派生类则还不能访问 _InternalField 成员
    }
}

```

6.4.1 private 私有的

使用关键字“private”描述的类型或类型成员是私有的，只能在类型内部访问，超出这个范围就不能访问了，派生类型也不能访问基类的私有成员。例如以下代码就定义了一个私有变量。

```
private string _PrivateField = null;
```

在这行代码中，关键字“private”说明这是私有的成员，“string”定义了成员的数据类型，“_PrivateField”为成员的名称，“=null;”定义了该字段的默认值。

6.4.2 protected 受保护的

使用关键字“protected”描述的类型或类型成员是受保护的，只有类型内部或从其派生的类型可以访问它，超出这个范围就不能访问了。以下代码就定义了一个受保护的字段。

```
protected string _ProtectedField = null ;
```

6.4.3 internal 内部的

使用关键字“internal”描述的类型和类型成员是程序集内部的，只能在程序集内部可见，其效果等价于“public”；而在程序集外部不可见，其效果等价于“private”。例如以下代码就定义了一个内部的字段。

```
internal string _InternalField = null;
```

在 VS.NET 中，由于一个程序集是由 C# 工程编译而得的，因此程序集内部就等价于 C# 工程中内部，程序集外部就是引用了该 C# 工程编译结果的其他 C# 工程。

例如当 C# 工程 A 中定义了一个 `internal` 类型时，在工程 A 中的任意 C# 代码都能访问这个内部类型；但当 C# 工程 B 引用了工程 A 的编译结果时，在工程 B 中的任意 C# 代码都不能访问工程 A 中定义的内部类型。

6.4.4 public 公开的

使用关键字 “`public`” 描述的类型或类型成员是公开的，在任何地方都能访问。以下代码就定义了一个公开的字段。

```
public string PublicField = null;
```

表 6-4 总结了可访问级别。

表 6-4 可访问级别

可访问级别	类型内部	派生的类型	同一个程序集的其他类型	不同程序集的其他类型
public	可见	可见	可见	可见
protected	可见	可见	非派生的不可见	非派生的不可见
Private	可见	不可见	不可见	不可见
internal	可见	同程序集的可见	可见	不可见

类型成员有可访问级别，而类型本身也有可访问级别，两者的效果一样。

当类型和类型成员的可访问级别不一致时，从类型外部看，对类型成员的可访问级别是类型的访问级别和类型成员自身的访问级别的叠加，取两者可见范围的交集。

例如当类型是 `public` 类型而某个成员方法是 `internal` 类型时，在程序集内部该成员方法到处都可见，但在程序集外部不可见。

6.5 类型样式

在定义类型的时候可以指定类型的一些特性，这些样式有如下几种。

6.5.1 static class 静态类

被 “`static`” 修饰的类就是静态类型，静态类型的所有成员都必须标记为静态的，否则会出现编译错误。静态类型一般用于容纳一些通用的例程，比如某些科学数值运算等。

以下 C# 代码就定义了一个静态类型。

```
public static class MyStaticClass
{
    public static int Sum(int a, int b)
    {
```

```

        return a + b;
    }
}

```

在代码“**public static class MyStaticClass**”中，“**public**”定义类型为公开的，“**static**”说明该类型是静态的，“**class**”说明正在定义一个类类型，“**MyStaticClass**”是类型名称。

在这个类型中定义了一个静态方法 **Sum**，在静态类型中不能定义非静态的成员。由于类型的构造函数也是静态的，因此静态类型不能实例化。

关于静态方法可参考 6.5.2 节。

6.5.2 abstract class 抽象类

被“**abstract**”修饰的类就是抽象类，抽象类是一种介于类和接口之间的类型，定义为抽象类说明其有部分内容尚未实现，有待以后被继承、被扩展。以下代码就定义了一个抽象类。

```

public abstract class MyAbstractClass
{
    public abstract int Sum(int a, int b);

    public int Div(int a, int b)
    {
        return a / b;
    }
}

```

在代码“**public abstract class MyAbstractClass**”中，关键字“**abstract**”声明了该类型为抽象类；在代码“**public abstract int Sum(int a, int b);**”中，关键字“**abstract**”声明该成员为抽象成员，抽象成员只能留个定义，不能有任何功能实现代码，因此在这里声明 **Sum** 方法就用分号结束了定义。

抽象类可以包含不抽象的方法，比如此处包含了一个完整的 **Div** 成员方法。

抽象类不能实例化，对于类型“我的抽象类”，代码“**MyAbstractClass instance = new MyAbstractClass()**”是错误的，抽象类必须派生出其他类型才能使用；而且派生的时候，所有的抽象成员必须强制被重写。由于抽象类必须被重载，所以抽象类不能是密封类，也就是说，关键字“**abstract**”和“**sealed**”是相互排斥的。

以下代码可从抽象类“**MyAbstractClass**”派生新的类型。

```

public class MyClassFromMyAbstractClass : MyAbstractClass
{
    public override int Sum(int a, int b)
    {
        return a + b;
    }
}

```

由于从抽象类派生新类型时，开发人员可能会忘记抽象类中定义的抽象成员，此时 VS.NET 的 C#代码编辑器会提供帮助功能，通过帮助能生成实现接口的功能类型，如图 6-4 所示。

```
public abstract class MyAbstractClass
{
    public abstract int Sum(int a, int b);

    public int Div(int a, int b)
    {
        return a / b;
    }
}

public class MyClassFromMyAbstractClass : MyAbstractClass
{
}
```

实现抽象类 “MyAbstractClass” (A)

图 6-4 自动生成实现抽象类的代码

当文本光标移动到方框处的 “MyAbstractClass” 时，左下角出现一个智能标签，用鼠标单击这个智能标签会弹出一个菜单，单击菜单项目 “实现抽象类 ‘MyAbstractClass’ ” 就会自动生成以下代码：

```
public class MyClassFromMyAbstractClass : MyAbstractClass
{
    public override int Sum(int a, int b)
    {
        throw new NotImplementedException();
    }
}
```

此时开发人员就可以往重载的成员中添加功能代码了。

由于抽象类型不能实例化，所以必须借助它的派生类型才能使用，例如以下代码：

```
MyAbstractClass instance = new MyClassFromMyAbstractClass();
instance.Div(1, 2);
instance.Sum(3, 4);
```

当然派生类型可以独立使用，例如以下代码：

```
MyClassFromMyAbstractClass instance2 = new MyClassFromMyAbstractClass();
instance2.Div(1, 2);
instance2.Sum(3, 4);
```

6.5.3 sealed class 密封类

被关键字 “sealed” 修饰的类是密封类，密封类可以继承自其他类，但不能被继承，不能包含虚方法和抽象方法。以下代码就定义了一个密封类。

```
public sealed class MySealedClass
{
    public int Sum(int a, int b)
    {
        return a + b;
    }
}
```

由于密封类不能被继承，不能被扩展，这可能会影响系统的扩展性，因此慎用，在实际中用得比较少。

6.6 类型成员

类型、接口和结构体都包含若干个类型成员，类型成员有多种。

例如以下代码就定义了一个类。

```
public class PeopleClass
{
    /// <summary>
    /// 静态构造函数
    /// </summary>
    static PeopleClass()
    {
        System.Console.WriteLine("Start");
    }

    /// <summary>
    /// 无参数的构造函数
    /// </summary>
    public PeopleClass()
    {
    }

    /// <summary>
    /// 有参数的构造函数
    /// </summary>
    /// <param name="code"></param>
    /// <param name="name"></param>
    public PeopleClass(string code, string name)
    {
        _Code = code;
        _Name = name;
    }

    /// <summary>
    /// 字段
    /// </summary>
    private string _Code = null;
    /// <summary>
    /// 属性
    /// </summary>
    public string Code
    {
        get
        {
            return _Code;
        }
        set
        {
            _Code = value;
        }
    }

    /// <summary>
    /// 字段
    /// </summary>
```

```
private string _Name = null;
/// <summary>
/// 属性
/// </summary>
public string Name
{
    get
    {
        return _Name;
    }
    set
    {
        _Name = value;
    }
}
/// <summary>
/// 方法
/// </summary>
/// <returns></returns>
public override string ToString()
{
    return _Code + " " + _Name;
}

private void RaiseNameChangeEvent()
{
    if (NameChanged != null)
    {
        NameChanged(this, null);
    }
}
/// <summary>
/// 事件
/// </summary>
public event EventHandler NameChanged = null;
}
```

这个类包含以下成员类型。

6.6.1 构造函数

在代码中定义了两个名为“PeopleClass”的方法，这种与类型同名的方法称为类型的构造函数。构造函数没有任何返回值，构造函数可以带参数，可以定义可用范围。

若在代码中没有定义任何构造函数，则系统会默认定义了一个无参数的公开的构造函数。

6.6.2 字段

在代码中使用“private string _Code = null;”定义了一个字段。在这里关键字“private”定义了该字段的引用范围为私有的，代码“string”说明了字段的数据类型，代码“_Code”为字段的名称，“=null”为字段的默认值。

6.6.3 属性

以下代码定义了一个属性。

```
public string Name
{
    get
    {
        return _Name;
    }
    set
    {
        _Name = value;
    }
}
```

在这里“**public**”定义了该属性的引用范围为公开的，代码“**string**”说明了属性的数据类型，“**Name**”说明了属性的名称，“**get**”代码块定义了如何获得属性值，“**set**”代码块定义了如何保存属性值，在“**set**”代码块中，关键字“**value**”是一个默认的参数，代表从外界传入的需要保存的数值。在 C# 中，只有在属性的“**set**”代码块中，关键字“**value**”才有效，其他地方出现的“**value**”必定是某个标识符，而不是关键字。

6.6.4 方法

以下代码定义了一个方法。

```
public override string ToString( )
{
    return _Code + " " + _Name;
}
```

在这里“**public**”定义了该方法为公开的；代码“**override**”说明该方法是重载了基类的相同签名的方法；“**string**”就是方法的返回值类型；“**ToString**”为方法的名称；圆括号“**()**”列表列出了方法的参数，这里内容为空表示该函数没有任何参数。

方法开头的后面就是方法体了，在里面使用“**return**”就是指退出本函数并返回一个字符串值。

方法有返回值和参数列表，返回值可以为任意数据类型，若函数没有返回值，则使用关键字 **void** 来定义方法，例如“**public void MyMethod()**”。有种说法，具有返回值的方法称为函数，没有返回值的方法称为方法。在本书中统一称为对象成员方法。

在定义方法具有返回值时，在方法体内的函数必须使用 **return** 语句返回运算结果，否则会出现编译错误，这点和 VB 是不同的。在 VB 中若没有使用 **return** 语句返回运算结果，则会自动返回与参数数据类型对应的默认值。

方法具有若干个参数，对于值类型的参数，C#默认是按照值传递的，但可以使用关键字 **out** 或 **ref** 来实现按引用传递；对于引用类型的参数，永远是按照引用传递的，例如“**public void MyMethod(ref int parameter)**”，如表 6-5 所示。

表 6-5 引用类型的参数

参数修饰符	说 明
无修饰	对于值类型的参数按值传递，在方法体内修改参数值不会影响方法体外的参数变量的值
out	说明该参数包含了方法的运算结果，此时参数是按引用传递的，而且在方法体内必须设置好参数值，否则会出现编译错误。在方法体内修改参数值会传导到方法体外用做方法产生的变量的值
ref	参数是按引用传递的。在方法体内修改参数值会传导到方法体外用做方法参数的变量的值。但方法体内不强制修改参数值，可以修改，也可以不修改

6.6.5 事件

以下代码定义了一个事件。

```
public event EventHandler NameChanged = null;
```

在这里“public”定义该事件为公开的；代码“event”说明这是定义一个事件；代码“EventHandler”就是事件采用的委托类型；代码“NameChanged”就是事件的名称；代码“=null”表示设置该事件的初始值为空。

本类型还定义了一个函数来触发这个事件，该函数的代码如下：

```
private void RaiseNameChangedEvent()
{
    if (NameChanged != null)
    {
        NameChanged(this, null);
    }
}
```

在这里代码“if (NameChanged != null)”用于判断事件是否有效，而代码“NameChanged(this , null)”就是调用该事件绑定的方法。

从便于理解的角度看，事件可以说是一种比较特殊的字段，它是一种列表，能存储若干个委托对象，从而能动态地绑定若干个方法。

6.6.6 索引器

索引器允许类或结构体的实例就像数组一样进行索引，索引器类似于一个带有参数的属性。以下代码定义了一个索引器。

```
/// <summary>
/// 索引器的范例
/// </summary>
public class PeopleList
{
    public PeopleList()
    {
    }

    private ArrayList _Values = new ArrayList() ;
    /// <summary>
```



```

    /// 添加对象
    /// </summary>
    /// <param name="people"></param>
    public void AddPeople(PeopleClass people)
    {
        _Values.Add(people);
    }

    /// <summary>
    /// 参数为序号的索引器
    /// </summary>
    /// <param name="index"></param>
    /// <returns></returns>
    public PeopleClass this[int index]
    {
        get
        {
            return ( PeopleClass ) _Values[index];
        }
        set
        {
            _Values[index] = value;
        }
    }

    /// <summary>
    /// 参数为人员姓名的索引器
    /// </summary>
    /// <param name="name"></param>
    /// <returns></returns>
    public PeopleClass this[string name]
    {
        get
        {
            foreach (PeopleClass people in _Values)
            {
                if (people.Name == name)
                {
                    return people;
                }
            }
            return null ;
        }
    }

    /// <summary>
    /// 根据生日计算年龄
    /// </summary>
    /// <param name="birthday"></param>
    /// <returns></returns>
    public static int CalculateAge( DateTime birthday )
    {
        return DateTime.Now.Subtract(birthday).Days / 365;
    }
}

```

在代码“**public PeopleClass this[int index]**”中，“**public**”说明该索引器是公开的，“**PeopleClass**”定义了该索引器使用的数据类型，“**this[]**”说明这是在定义一个索引器，“**int index**”说明该索引器有个名为“**index**”类型为“**int**”的参数。

在代码“`public PeopleClass this[string name]`”中，“`string name`”说明该索引器使用的是一个字符串类型的参数，这个索引器可以按照人员姓名来获得人员信息对象。

在索引器的“`get`”代码块中，程序根据“`index`”参数值获得返回值；在“`set`”代码块中，程序根据“`index`”参数值和默认存在的“`value`”参数值来设置内部数值。

若索引器只有“`get`”代码块而没有“`set`”代码块，则该索引器是只读的，不能通过它来修改数据；若只有“`set`”代码块而无“`get`”代码块，则该索引器只能用于修改数据而不能用于获得数据。

以下 C# 代码就演示了如何使用索引器。

```
//创建对象
PeopleList list = new PeopleList();
//向对象添加内容
list.AddPeople(new PeopleClass("1", "张三"));
list.AddPeople(new PeopleClass("2", "李四"));
list.AddPeople(new PeopleClass("3", "王五"));
list.AddPeople(new PeopleClass("4", "陈六"));
list.AddPeople(new PeopleClass("5", "赵七"));
list.AddPeople(new PeopleClass("6", "钱八"));

//使用索引器获得数据
PeopleClass people = list[0];

people = list["赵七"];
```

从这段代码中可以看出，带有索引器的类型的对象实例可以当做数组来使用，可以使用下标来访问它的数据。

索引器比数组更为灵活，因为数组只能使用整数作为下标值，而索引器则可以使用任何数据类型作为下标值。例如在这里使用代码“`public PeopleClass this[string name]`”定义的索引器就可以使用字符串作为下标值。此外索引器可以有多个参数来支持多个下标值。

索引器可以被重载，可以定义抽象索引器或虚拟索引器。

6.6.7 静态成员

使用关键字“`static`”修饰的类型成员就是静态成员，类型的字段、属性、方法、事件、构造函数都可以是静态的。例如在上面的 `PeopleClass` 中使用以下代码定义了一个静态方法。

```
public static int CalculateAge( DateTime birthday )
{
    return DateTime.Now.Subtract(birthday).Days / 365;
}
```

这个静态方法能根据生日计算周岁。由于计算周岁是一个通用的算法，并不局限于某个特定的对象，因此可以将其定义为静态类。

静态成员类型无须创建对象实例即可通过“类型名称.成员名”的方式调用。比如以下代码就能调用 `CalculateAge` 方法。

```
DateTime dtm = new DateTime(1980, 2, 14);
int age = PeopleClass.CalculateAge(dtm);
```

对于静态类型，通过“实例变量.成员名”的方式反而不能调用，比如以下代码就是错误的。

```
PeopleClass instance = new PeopleClass();
DateTime dtm = new DateTime(1980, 2, 14);
int age = instance.CalculateAge(dtm);
```

注意，调用静态成员在 C# 和 VB.NET 语法中有些差别。在 VB.NET 中，可以使用“派生的类型名称.成员名”来调用静态成员，但 C# 不支持。例如，在类型 A 中定义了静态成员 M，而从类型 A 之上派生了类型 B，则在 VB.NET 中可以使用“A.M”或“B.M”来调用静态成员，但在 C# 中不能用“B.M”这种写法。

以下代码中定义了静态方法“Sum”和静态字段“Value”。

```
public class MyClass
{
    public static int Value = 100 ;
    public static int Sum(int a, int b)
    {
        return a + b;
    }
}
```

这样我们就可以使用“MyClass.Sum”来调用这个方法而无须创建对象实例。若还从这个类型派生了新类型，其代码如下：

```
public class OtherClass : MyClass
{
}
```

则在 C# 中只能使用“MyClass.Sum”来调用这个静态方法，而在 VB.NET 中可以用“MyClass.Sum”和“OtherClass.Sum”两种方式来调用这个静态方法。

1. 静态字段

对于静态字段，是可以赋值的，而且静态字段作用范围是整个程序，相当于全局变量。例如，在任意地方修改了此处的静态变量“Value”的值后，其他地方获得的字段值就是修改后的值。

2. 静态构造函数

被“static”修饰的无参数构造函数就是类型的静态构造函数。在上面的 PeopleClass 类型中就使用以下代码定义了一个静态构造函数。

```
static PeopleClass()
{
    System.Console.WriteLine("Start");
}
```

在程序加载的时候，若程序没有调用 PeopleClass 中的成员，则静态构造函数不会被调用，甚至使用 typeof 操作也不会调用；但程序中第一次引用类型的某个成员，或者创建对象实例前，系统会自动地调用一次类型的静态构造函数，而且在整个软件运行期间，某个类型的静态构造函数

数只可能被调用一次，是不会被重复调用的。

静态构造函数适用于进行系统初始化的延时操作，这能加快系统的启动速度；而且在程序的整个运行过程中，可能没有调用静态构造函数，这样能降低程序的工作量，提高代码的执行效率。

注意，静态构造函数必须是没有参数的，而且肯定是私有的，不能设置可访问级别。

6.6.8 实例成员

没有被“static”关键字修饰的类型成员就是实例成员，例如在 `PeopleClass` 类型中定义了一个 `ToString` 方法，其代码如下：

```
public override string ToString()
{
    return _Code + " " + _Name;
}
```

此时不能用“`PeopleClass.ToString`”来调用这个方法，必须首先创建对象实例，然后再调用这个对象实例的方法，其演示代码如下：

```
PeopleClass instance = new PeopleClass( );
instance.ToString( );
```

1. 虚拟成员

被关键字“virtual”修饰的成员类型是虚拟成员，例如以下代码就包含了一个名为“Sum”的虚拟函数。

```
public class MyClass
{
    public virtual int Sum(int a, int b)
    {
        return a + b;
    }
}
```

其实，虚拟成员并不虚拟，它可以包含实质性的功能代码，能完成一定的功能。只不过虚拟成员可以方便地被重载而已，继承者也可以根据需要不重载这些虚拟方法。

2. 抽象成员

被关键字“abstract”修饰的类型成员就是抽象成员。抽象成员只能是属性、方法和索引器。以下代码就定义了一个抽象方法。

```
public abstract int Sum(int a, int b) ;
```

在这段代码中，“public”说明方法是公开的；“abstract”说明这是一个抽象的成员；“int”为方法的返回值类型；“Sum”为方法的名称；“int a, int b”为方法的参数列表。

定义抽象类型和定义接口的方法类似，需要写出成员的声明，另外还需要写出抽象成员的可访问性。

抽象成员必须出现在抽象类中，而且在从抽象类派生新类型时，所有的抽象方法必须重写以填充方法体。

抽象成员和虚拟成员是有区别的，抽象成员不能定义任何实质的功能，必须被重载；而虚拟成员必须包含完整的代码结构，可以包含实质功能，可以不被重载。

6.6.9 常数成员

被关键字“const”修饰的成员字段为常数字段，可以使用“类型名称.字段名”来引用常数。例如以下代码就包含了一个常数。

```
public class MyClass
{
    public const double PI = 3.1415926;
}
```

对此我们可以使用“MyClass.PI”来获得这个常数值。常数值是不能修改的，比如对于赋值代码“MyClass.PI = 3.14;”就是错误的。相对地，静态字段是可以被修改的，比如此处使用代码“public static double PI = 3.1415926;”来定义这个字段，就可以对这个字段赋值了。

6.7 面向对象编程

C#是完全的面向对象的编程语言，关于面向对象编程思想可在相关章节阅读，本章只说明C#是如何实现面向对象编程思想的。

6.7.1 类

在C#中，没有游离于类型之外的变量和方法，所有变量和方法都属于某个类型，在C#中最多的类型就是类类型。

6.7.2 封装

在类中可以定义字段，字段默认是私有的，也应该是私有的，比如以下代码定义了一个类。

```
public class PeopleClass
{
    private string _Name = null;
    public string Name
    {
        get
        {
            return _Name;
        }
        set
        {
            _Name = value;
        }
    }
}
```

有人可能说，上面的代码太复杂，那么可以写出下面的代码，功能一样。

```
public class PeopleClass
{
    public string Name = null ;
}
```

笔者推荐第一种写法，字段设置为私有的，然后使用公开的属性来访问字段值。关于封装的必要性可参考后面的相关章节。

6.7.3 继承

C#支持对象的成员方法、属性和索引器的继承。派生的类型继承了基类所有公开或受保护的成员。

以下代码定义了一个“CustomerClass”类型，该类型是从“PeopleClass”派生的。

```
public class CustomerClass : PeopleClass
{
    private string _CompanyName = null;
    public string CompanyName
    {
        get
        {
            return _CompanyName;
        }
        set
        {
            _CompanyName = value;
        }
    }
}
```

此时“CustomerClass”除了拥有自己的“CompanyName”属性外，还从 PeopleClass 类型处继承了 Name 属性。也就是说，以下代码是有效的。

```
CustomerClass instance = new CustomerClass();
instance.Name = "张三";
instance.CompanyName = "蓝天公司" ;
```

6.7.4 重载

重载就是在一个类型中定义多个类型成员，这些成员具有相同的名字、不同的签名。重载最常见的就是成员方法重载。

以下代码定义了三个类型成员方法。

```
public int Sum(int a, int b)
{
    return a + b;
}
public int Sum(int a, int b, int c)
{
    return a + b + c;
}
public double Sum(double a, double b)
```

```
{
    return a + b;
}
```

这三个方法名称相同，但具有不同的参数签名，也就是说，参数类型、参数个数不完全一样。通过签名重载，这里的 `Sum` 方法有 3 个重载版本。

在调用的时候，系统会根据参数来确定使用哪个版本的方法。例如执行以下代码：

```
int a = Sum(2, 3);
int b = Sum(4, 5, 6);
double c = Sum(4.3, 5.1);
```

对于代码 “`Sum(2 , 3)`”，由于使用了两个整数参数，因此调用了 `Sum` 方法的第一个版本；对于 “`Sum(4 , 5, 6)`”，由于使用了三个整数参数，因此调用了 `Sum` 方法的第二个版本；对于 “`Sum(4.3, 5.1)`”，由于使用了两个浮点数参数，因此调用了 `Sum` 方法的第三个版本。

使用签名重载，能增加程序的可读性和可维护性。

6.7.5 重写

派生类型能重写基类型中的一个属性、索引器和方法的功能代码，在保持接口不变的情况下对外提供新的功能，而外界对此并不知晓，还照常调用。因此重载是一种对软件架构偷梁换柱的手法。

例如以下代码：

```
public abstract class 抽象的变形金刚
{
    public virtual string 变形()
    {
        return "啥都不是";
    }
}

public class 擎天柱:抽象的变形金刚
{
    public override string 变形()
    {
        return "卡车头";
    }
}

public class 大黄蜂:抽象的变形金刚
{
    public override string 变形()
    {
        return "黄色跑车";
    }
}

public class 红蜘蛛 : 抽象的变形金刚
{
    public override string 变形()
    {
        return "战斗机";
    }
}
```

```
    }  
}
```

在这段代码中，定义了一个抽象类型“抽象的变形金刚”，它有一个虚拟的方法“变形”，返回一个变形结果；而其他类型“擎天柱”、“大黄蜂”和“红蜘蛛”都派生自“抽象的变形金刚”，而且重载了“变形”方法，实现了各自具体的效果。

以下代码就测试使用了这些类型。

```
抽象的变形金刚 man = new 擎天柱();  
  
Console.WriteLine( man.变形()); //输出"卡车头"  
  
man = new 大黄蜂();  
Console.WriteLine(man.变形()); //输出"黄色跑车"  
  
man = new 红蜘蛛();  
Console.WriteLine(man.变形()); //输出"战斗机"
```

这里定义了变量 `man`，类型为“抽象的变形金刚”，并将其指向不同的具体类型的变形金刚对象实例，若调用它的“变形”方法，则会输出不同的值。

通过继承重载，在保持接口不变的情况下，程序模块实现了动态的效果，这能让应用系统达到很高的灵活性。

6.8 表达式

C# 语法支持非常丰富的表达式，包括数学表达式、逻辑表达式和位运算表达式。

6.8.1 数学表达式

C# 的数学表达式支持加、减、乘、除和求模运算，与其他编程语言差不多。其支持的运算符如表 6-6 所示

表 6-6 C# 支持的数学运算符

运 算 符	说 明
+	加，例如“2+3”，运算结果为 5
-	减，例如“2-3”，运算结果为-1
*	乘，例如“2*3”，运算结果为 6
/	除，例如“2/3”，运算结果为“0.66666”
%	求模，例如“2%3”，运算结果为“2”
++	自增 1，例如“int a = 2; a++”，运算结果为“3”
—	自减 1，例如“int a = 2; a—”，运算结果为“1”

C# 允许使用圆括号来改变运算的优先级。

6.8.2 逻辑表达式

C#支持的逻辑表达式与其他编程语言差不多，其支持的运算符如表 6-7 所示。

表 6-7 C#支持的逻辑运算符

意 义	优 先 级	语 法
逻辑反	高	!
等于	中	== （两个等于号，中间不能有空格）
大于	中	>
大于等于	中	>=
小于	中	<
小于等于	中	<=
不等于	中	!=
逻辑与	低	&&
逻辑或	低	

特殊逻辑运算符如表 6-8 所示。

表 6-8 特殊逻辑运算符

运 算 符	说 明
&&	逻辑与，比如 “BooleanValue1 && BooleanValue2”，只有两个条件都成立，这个表达式才成立
	逻辑或，比如 “BooleanValue1 BooleanValue2”，两个条件中任意一个成立，则这个表达式就会成立
!	逻辑反，比如 “! BooleanValue”，若条件成立，则表达式不成立，若条件不成立，则表达式成立

C#允许使用圆括号来改变运算的优先级。

6.8.3 位运算表达式

C#支持类似 C 语言的位运算，如表 6-9 所示。

表 6-9 C#支持的位运算符

运 算 符	说 明
<<	左位移，相当于乘以 2 的若干次方，比如 “int a = 3 ; a <<2”，其效果等于 “a = a * 2 * 2”，运算结果为 12。不过左位移操作比乘法操作要快得多，当固定为乘上 2 的若干次方值时，尽量用左位移操作
>>	右位移，相当于除以 2 的若干次方，比如 “int a = 100 ; a >>2”，其效果等于 “a = a / 4”，运算结果为 25。不过右位移操作比除法操作要快得多，当固定为除以 2 的若干次方值时，尽量用右位移操作
&	位与操作
	位或操作

6.9 执行结构

C#采用的是类似 C 语言的执行结构，包括顺序执行和条件判断。

6.9.1 顺序执行

顺序执行就是代码一行一行地执行，一直到结束。

6.9.2 条件判断

在 C#中有多种条件判断执行语句，主要有以下几种结构。

1. if 结构

C#支持 if 条件判断结构，使用很简单，其演示代码如下：

```
DateTime dtm = DateTime.Now;
string b = null;
if (dtm.Hour >= 12)
{
    b = "下午";
}
else
{
    b = "上午";
}
```

理论上这段代码中的花括弧可以去掉，写成：

```
DateTime dtm = DateTime.Now;
string b = null;
if (dtm.Hour >= 12)
    b = "下午";
else
    b = "上午";
```

注意，在实践中是不推荐这种写法的。这种写法虽然比较简练，但很可能制造出难于察觉的程序错误，还应该老老实实地把花括号写出来。

在书写 if 语句时，若逻辑表达式是复合的，包含了多个条件，则需要换行。例如对于以下代码，if 语句中包含了一个具有 4 个成员的复合逻辑表达式。

```
int NumberValue = 10;
string StringValue = "123";
char CharValue = 'z';
double DoubleValue = 30;

if ( NumberValue == 10 || StringValue == "123" || CharValue == 'z' ||
DoubleValue >= 20 )
{
    Console.WriteLine("条件成立");
}
```

这种写法不美观，一条语句过长，推荐写成如下样式：

```
int NumberValue = 10;
string StringValue = "123";
char CharValue = 'z';
double DoubleValue = 30;

if ( NumberValue == 10
    || StringValue == "123"
    || CharValue == 'z'
    || DoubleValue >= 20 )
{
    Console.WriteLine("条件成立");
}
```

另外，如果判断条件就是一个布尔值，理论上可以写成如下代码：

```
bool BooleanValue = true;
if ( BooleanValue )
{
    Console.WriteLine("条件成立");
}
```

但在实践中，还是推荐写成如下代码：

```
bool BooleanValue = true;
if ( BooleanValue == true )
{
    Console.WriteLine("条件成立");
}
```

虽然上一种写法比较简短，但第二种写法的可读性更好。

类似地，在判断这个布尔值是否为 **false** 时，可以使用逻辑反运算符，其代码如下：

```
bool BooleanValue = true;
if ( ! BooleanValue )
{
    Console.WriteLine("条件成立");
}
```

这种写法的可读性也不好，因为人们在快速阅读代码时有可能将“!”这个字符漏掉，因此在实践中，推荐写成如下代码：

```
bool BooleanValue = true;
if ( BooleanValue == false )
{
    Console.WriteLine("条件成立");
}
```

在实际编写代码时，代码的可读性比简短要重要，应优先考虑。笔者反对编写简短隐晦的代码来炫耀技术，这与孔乙己说“茴”字有 4 种写法一样不切实际。

C#中的三元运算符代码结构为“判断条件 ? 当判断成立时的值 : 当判断不成立时的值”。例如以下代码演示了三元运算符。

```
DateTime dtm = DateTime.Now;
string b = dtm.Hour >= 12 ? "下午" : "上午";
```

在这段代码中，若当前时间的小时数大于等于 12，则获得数值“下午”，否则获得数值

“上午”。

这种三元运算符功能和 if-else 语句一样，能以更为简短的方式实现功能。在很多情况下能减少代码量，提高代码的可读性。不过三元运算符没有扩展性，一次只能设置一个变量值。

2. switch 结构

C# 中的 switch 结构为 “if (判断条件) 执行代码 ; else 执行代码”，比如以下代码说明了作息状态。

```
DateTime dtm = DateTime.Now;
string state = null;
switch (dtm.Hour)
{
    case 6:
    {
        state = "起床";
    }
    break;
    case 8:
    {
        state = "吃早饭";
    }
    break;
    case 9:
    {
        state = "上班";
    }
    break;
    case 12:
    {
        state = "吃午饭";
    }
    break;
    case 13:
    {
        state = "上班";
    }
    break;
    case 18:
    {
        state = "下班";
    }
    break;
    case 22:
    {
        state = "睡觉";
    }
    break;
}
Console.WriteLine(state);
```

其中的 switch 语句块还可以写成如下的紧凑样式：

```
switch (dtm.Hour)
{
    case 6: state = "起床"; break;
```

```

        case 8: state = "吃早饭"; break;
        case 9: state = "上班"; break;
        case 12: state = "吃午饭"; break;
        case 13: state = "上班"; break;
        case 18: state = "下班"; break;
        case 22: state = "睡觉"; break;
    }

```

若使用 **if-else** 结构写出这种语句，则可以写成：

```

DateTime dtm = DateTime.Now;
string state = null;
if (dtm.Hour == 6)
{
    state = "起床";
}
else if (dtm.Hour == 8)
{
    state = "吃早饭";
}
else if (dtm.Hour == 9)
{
    state = "上班";
}
else if (dtm.Hour == 12)
{
    state = "吃午饭";
}
else if (dtm.Hour == 13)
{
    state = "上班";
}
else if (dtm.Hour == 18)
{
    state = "下班";
}
else if (dtm.Hour == 22)
{
    state = "睡觉";
}
Console.WriteLine(state);

```

可以看出对于多个常数的比较判断，使用 **switch** 结构还是比较方便的。但在 **case** 语句中只能是常数，不能是逻辑表达式，这使得 **switch** 结构的应用受到不小的限制。

在 **switch** 语句块中可以不使用花括号，因为不添加花括号不会导致难于察觉的错误。但推荐若 **case-break** 结构中的功能代码比较多，则加花括号。

在 C 语言中，**case** 结构如果不加上 **break** 关键字就可以贯穿执行到下一个 **case** 结构。但在 C# 中，**case** 结构不能贯穿，必须添加 **break** 或 **return** 关键字来结束 **case** 执行块。

6.9.3 循环结构

在 C# 中有多种循环结构，以下分别说明。

1. for 循环结构

在 C# 中，for 循环结构语法为 “for(初始化 ; 逻辑表达式 ; 累加处理){ 循环体 }”。以下代码就包含了一个 for 循环结构。

```
int count = 0;
for (int index = 0; index < 100; index++)
{
    count = count + index;
}
Console.WriteLine(count.ToString());
```

理论上这段代码可以写成：

```
int count = 0;
for (int index = 0; index < 100; index++)
    count = count + index;
Console.WriteLine(count.ToString());
```

但笔者推荐还是加上花括号，因为不加花括号很容易导致难以察觉的程序错误。

在代码 “for(int index = 0 ; index < 100 ; index ++)” 中，关键字 “for” 说明开始进行了一个循环；“int index = 0” 表示初始化这个循环，定义了一个累加变量；“index < 100” 是这个循环是否执行的逻辑表达式，若判断成立则执行循环，否则立即退出循环；“index ++ ” 是每次执行完一次循环体后执行的代码，这里是将累加变量自增 1。

上面 for 循环执行的步骤如下：

```
int count = 0;
int index = 0;
//执行第 1 次循环
if (index < 100)
{
    count = count + index;
    index++;
}
else
{
    goto EndFor;
}
//执行第 2 次循环
if (index < 100)
{
    count = count + index;
    index++;
}
else
{
    goto EndFor;
}
//执行第 3 次循环
if (index < 100)
{
    count = count + index;
    index++;
}
else
```

```

    {
        goto EndFor;
    }

    //无穷尽地执行循环体

EndFor:
    Console.WriteLine(count.ToString( ));

```

注意，理论上循环会无穷尽地执行下去，因此需要设置好退出循环的判断条件，否则容易出现死循环，导致死机。

for 循环是使用最为灵活的循环结构，在实践中可以使用它写成很多精致的代码。

在各种循环结构中，包括 for 循环，都可以使用关键字 **continue** 来跳过一些代码直接进入下一次循环，使用关键字 **break** 来立即退出循环。若从套嵌了多层的循环中跳出来，则可以使用 **goto** 语句。

2. foreach 循环结构

foreach 循环结构是一种针对列表类型的循环语法结构，其语法为“**foreach**(类型名称 变量名 **in** 列表变量名){ 循环体 }”，以下代码演示了 **foreach** 的使用。

```

string[ ] names = new string[100];
foreach (string name in names)
{
    Console.WriteLine(name);
}

```

在 **foreach** 循环中，循环变量是只读的，不能为它设置值，比如上面的 **foreach** 循环体中不能写上“**name**=“张三”；”这样的代码，但 **for** 语句是可以的。

foreach 循环本质上是 C# 的一个语法封装，上述这段代码等效于如下代码：

```

string[ ] names = new string[100];
System.Collections.IEnumerator enumerator
    = ((System.Collections.IEnumerable)names).GetEnumerator();
enumerator.Reset();
while (enumerator.MoveNext())
{
    string name = (string)enumerator.Current;
    Console.WriteLine(name);
}

```

因此任何类型只要实现了“**System.Collections.IEnumerable**”接口的类型就可以用于 **foreach** 循环结构。这段代码中“**enumerator.Current**”是只读的属性，这就是为什么 **foreach** 循环体内不能设置循环变量的值的原因。

3. while 循环结构

C# 执行 **while** 循环结构，其代码结构为“**while**(逻辑表达式) { 循环体 }”，例如前面的演示程序：

```

string[ ] names = new string[100];

```

```
System.Collections.IEnumerator enumerator
    = ((System.Collections.IEnumerable)names).GetEnumerator();
enumerator.Reset();
while (enumerator.MoveNext())
{
    string name = (string)enumerator.Current;
    Console.WriteLine(name);
}
```

就是一个 **while** 循环结构。这个循环等价于执行以下代码：

```
//执行第一次循环
if (enumerator.MoveNext())
{
    string name = (string)enumerator.Current;
    Console.WriteLine(name);
}
else
{
    goto EndWhile;
}
//执行第二次循环
if (enumerator.MoveNext())
{
    string name = (string)enumerator.Current;
    Console.WriteLine(name);
}
else
{
    goto EndWhile;
}
//执行第三次循环
if (enumerator.MoveNext())
{
    string name = (string)enumerator.Current;
    Console.WriteLine(name);
}
else
{
    goto EndWhile;
}
//无穷尽地执行循环
EndWhile:
```

4. do-while 循环结构

C# 支持 **do-while** 循环结构，其语法为 “do{ 循环体 }while (逻辑表达式);”。以下代码演示了 **do-while** 循环结构。

```
string[] names = new string[100];
int index = 0;
do
{
    Console.WriteLine(names[index]);
    index ++ ;
}
while (index < names.Length - 1);
```

这段代码相当于执行以下代码：


```

//执行第一次循环
Console.WriteLine(names[index]);
index++;
if (! (index < names.Length - 1))
{
    goto EndDoWhile;
}
//执行第二次循环
if (index < names.Length - 1)
{
    Console.WriteLine(names[index]);
    index++;
}
else
{
    goto EndDoWhile;
}
//执行第三次循环
if (index < names.Length - 1)
{
    Console.WriteLine(names[index]);
    index++;
}
else
{
    goto EndDoWhile;
}
//无穷尽地执行循环
EndDoWhile:

```

从这段代码可以看出，do-while 循环和 for、while 循环的不同之处是它在执行第一次循环体前不计算逻辑表达式，让代码无保障地运行在高风险区，使之成为一个程序错误的高发区。因此笔者很少使用 do-while 语句，只在某些特殊的情况下才使用。

笔者建议，要写出稳定健壮的程序，开发人员必须有很强的风险意识，对意外不能有侥幸心理，认认真真地尽量处理所有可能出现的风险。对一个程序应该编写大量的代码来确保其有一个安全的环境，让功能性代码的运行得到保障。

6.9.4 异常处理结构

开发软件时一定要有足够的风险意识，认识到商业软件在各种复杂的情况下运行，必然会出现各种各样的风险和错误，这些风险和错误需要进行处理。无视风险和错误，假设一切都很和谐是很危险的思想。

1. 主动处理错误

程序开发中可以主动处理错误，也可以被动处理错误，主动处理错误就是写代码和做功能执行前的检查，最常见也是最有效的手段就是在方法体开头处检查方法参数是否正确。主动检查程序不仅能使运行速度快，而且系统更稳定，能将风险消灭在萌芽之中，避免后期的错误大爆发，因此是优先采用的风险处理方式。

下面的代码就是在主动处理错误。

```
public int Div( int a , int b )
{
    //检查参数，若 b 为 0，则必然会报被 0 除的错误，提前处理错误
    if (b == 0)
    {
        return 0;
    }
    return a / b;
}
```

这段代码就是在主动地检查参数是否正确，尽早制止错误的发生。

2. 被动异常处理

主动处理错误的方式很可能会有遗漏，C#中可以采用异常捕获机制来被动地处理错误。

在说明异常前首先介绍程序中调用堆栈的概念。其实基本上所有编程语言都有调用堆栈的功能。例如在程序运行时，方法 F1 调用了方法 F2，F2 在某个时刻又调用了方法 F3，而方法 F3 在某个时刻调用了方法 F4，则此时存在“F1 | F2 | F3 | F4”的调用堆栈。如图 6-5 所示，此时程序的代码呈现为一种按照方法分层的“洋葱”结构，越靠里层就越是底层代码。

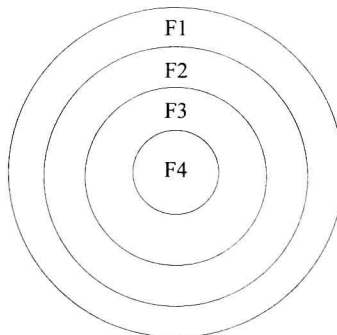


图 6-5 按照方法分层的“洋葱”结构

C#中的异常是指程序模块发出错误的信息，抛出异常是指程序通知系统程序发生错误了，这种异常是全局性的，从抛出点开始，沿着调用堆栈，一层层地向上推动，越往上，影响面越大。若调用堆栈上某层代码能主动捕获并处理这个异常，该异常就会消失，系统就能正常运行。若一直没有代码能主动捕获异常，则该异常就会一直捅到.NET 框架层面。

例如，在 F1、F2、F3、F4 构成的调用堆栈中，如果方法 F4 内部主动抛出一个异常，该异常首先抛给方法 F3，若 F3 不处理异常则该异常又抛给了 F2，若 F2 还是没有处理则继续抛给了 F1，若 F1 没有处理则直接抛给了.NET 框架系统本身。

.NET 框架就是.NET 应用程序的中央，.NET 框架处理没有被应用程序处理的异常会弹出如图 6-6 所示的程序错误对话框。

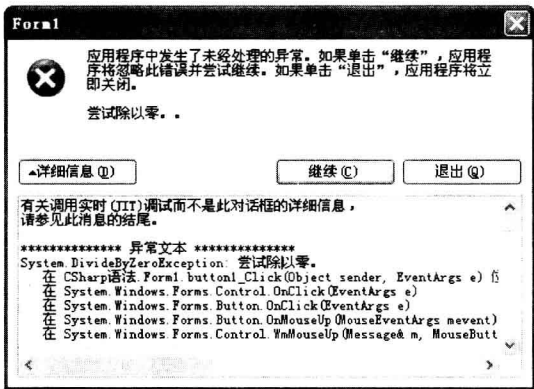


图 6-6 程序错误对话框

在该对话框中，若单击“继续”按钮则程序或许还能接着运行，但可能处于一种不稳定的状态；若单击“退出”按钮则立即无条件地退出程序，这可能导致数据未保存。

一旦显示出系统级错误对话框，则该程序的稳定性和可靠性都是非常差的，应当尽量避免这种情况。这就需要在调用堆栈中的某些合适层面的方法中添加异常处理，来应付底层方法抛出的异常。

在 C#中使用“throw”关键字抛出异常，以下代码就演示了如何抛出异常。

```
public int F1()
{
    throw new Exception("这里发生异常了");
}
```

在这个方法中使用 throw 关键字主动抛出了一个异常。在 C#中所有异常都是某个底层方法主动抛出来的。

注意，方法 F1 定义为需要返回一个整数数值，若方法体中没有 return 语句编译不通过的，则使用 throw 语句可以压住这个规定。

在 C#中使用“try{ }catch{ }finally{ }”结构来处理异常，以下代码就演示了如何处理异常。

```
public void HandleException(int a, int b)
{
    try
    {
        //进入能捕获异常的状态
        F1();
    }
    catch (Exception ext)
    {
        //若发生异常，在此捕获异常，执行这段代码
        System.Windows.Forms.MessageBox.Show(ext.Message);
    }
    finally
    {
        //不管有没有发生异常，本段代码都运行
        Console.WriteLine("完成了处理");
    }
}
```

```
}  
}
```

在关键字“try”后面的代码块一般放置方法的功能性代码，此时系统时刻检查是否抛出异常。若抛出异常则立即捕获它，然后跳转到关键字“catch”后面的代码块中处理异常。

这段代码中，无论是否发生异常，关键字“finally”后面的代码块都会执行，这段代码一般用于本方法的善后工作，比如释放在工作过程中申请的重要资源等。catch 块和 finally 块是可选的，也可以写成“try{ } catch{ }”或“try{ } finally{ }”。

对于“try{ } catch{ }”结构，程序会捕获异常并将其处理掉，使得异常不再上访。

对于“try{ } finally{ }”结构，程序会感知异常，做一些处理，但不会捕获异常，异常仍然会接着上访。

使用异常处理，就能让底层方法抛出的异常被及时地处理掉而不至于一路捅到.NET 框架，进而成为系统级的错误，因此开发程序时必须加上合适的异常处理。

由于异常是被动地处理错误，而且执行过程消耗了不少资源，因此主动防御错误永远比被动处理错误要好。在实践中可以考虑将两者都用上，做好双保险，这样程序才能可靠稳定。

3. using 语法结构

在 C# 中，关键字 using 除了可以引用命名空间外，还可以使用 using 结构来实现一定的异常处理。其语法结构为“using() { }”。以下代码就演示了 using 结构。

```
using (System.IO.FileStream stream  
    = new System.IO.FileStream("c:\\a.txt", System.IO.FileMode.Create))  
{  
    stream.WriteByte((byte)2);  
}
```

在字段代码中，“using”后面的圆括号中为打开文件创建了一个文件流对象，然后在后面的花括号的代码块中向这个文件输出一字节，就没有其他代码了。

文件流是一种很重要的资源，打开后必须关闭，否则就会将资源泄露，在上面的代码中应该补上代码“stream.Close()”来关闭文件流。但实际上上面的代码是安全的，不会出现资源泄露问题，这是因为采用了 using 语法结构。

上述代码其功能等价于以下代码：

```
System.IO.FileStream stream  
    = new System.IO.FileStream("c:\\a.txt", System.IO.FileMode.Create);  
try  
{  
    stream.WriteByte((byte)2);  
}  
finally  
{  
    ((System.IDisposable)stream).Dispose();  
}
```

C#编译器会将 using 结构翻译成一个“try{ } finally{ }”结构，并在 finally 块中调用文件流

对象的 `Dispose` 方法，这个方法能关闭文件流，释放所有资源。即使其中的功能代码发生错误，抛出异常，文件流仍然能关闭以便释放资源。

从这里可以看出 `using` 语法结构具有一定的异常处理功能，它没有捕获异常，但能感知异常，并进行一些处理来减少异常带来的负面影响。

C#中的 `using` 语法结构是针对 `System.IDisposable` 接口的，`System.IDisposable` 接口只声明了一个成员 “`void Dispose()`”，用于释放对象所占有的重要资源。任何实现了 `System.IDisposable` 接口的对象都能被用上 `using` 结构，比如文件流、数据库连接、网络连接等。若一个类型没有实现 `IDisposable` 接口而使用 `using` 语法结构，则会报编译错误。

6.10 C#转型建议

由于一些读者已经掌握了其他编程语言的技术，所以对此类读者提供一些 C#转型建议，希望能帮助这类读者更好地掌握 C#语言。

6.10.1 从 VB 到 C#

笔者就是从 VB 6.0 转变到 C#的，在此提出一些 VB 转变到 C#的建议，希望能给那些已经掌握 VB 的读者一些帮助。

C#的语法采用 C 语言样式，而且其设计大量参考了 VB 的思想。其转变要点如下：

(1) VB 的变量、方法和类型的名称是不区分大小写的，而 C#是区分大小写的。

(2) VB 的语句是使用换行来确定的，而 C#是用分号 “;” 来分隔语法单位的。

(3) VB 进行复合条件判断时，所有的条件项目都会执行判断，而 C#是依次执行的，若遇到确定情况则立即结束判断。

例如在 VB 中执行语句 “`if value1 <> 0 or value2 = 3 then`”，若判断出 “`value1 <> 0`” 成立后，按理来说无论第二个条件是否成立，这个复合条件都是成立的，但在 VB 中程序还会计算 “`value2 = 3`” 是否成立，当所有的条件项目判断完毕后才得出整个复合判断条件是否成立。

而在 C#中执行等价的语句 “`if(value != 0 or value2 == 3)`”，若判断出 “`value1 != 0`” 成立，则程序认定这个复合判断条件成立，不再计算 “`value2 == 3`” 的判断条件，避免无谓的运算量。

从这个过程可以看出，C#在执行复合条件判断时比 VB 效率高。

- `foreach` 语句。

C#支持 VB 中的 `foreach` 语句，VB 中的语法是 “`foreach 变量名 in 列表变量名`”，而 C#的语法是 “`foreach(类型名称 变量名 in 列表变量名)`”。

例如有一个整数数组，在 VB 中遍历数组的写法是：

```
Dim array(10) as Integer
```

```
Dim item as Integer
For Each item in Array

Next
```

而 C# 的写法是：

```
int[] array = new int[10];
foreach( int item in array )
{
}
```

- 数组下标

VB 的数组下标默认是从 1 开始计算的，而 C# 是从 0 开始计算的；VB 的 UBound 函数返回的是可用下标的最大值，而 C# 的数组对象的 Length 或 Count 属性返回的是数组中元素的个数，此时可用下标的最大值是要减去 1 的。

例如，在 VB 中使用语句“Dim Array(10) As Integer”定义一个数组，则它的元素个数是 10，最小下标是 1，最大下标是 10。

而在 C# 中使用语句“int[] Array = new int[10]”定义的数组，它的元素个数是 10，最小下标是 0，最大下标是 9，而它的 Length 属性值为 10。

在 VB 中使用数组下标时，下标值两边是圆括号，但在 C# 中使用方括号。例如 VB 代码“Array(9)”就是指获得数组中第 9 个元素的数值，而在 C# 中必须写成“Array[9]”。

- 参数传递方式

C# 和 VB 在进行函数参数的传递时都有按值传递和按引用传递的区分。VB 的函数参数值默认是按引用传递的，而 C# 的函数参数值默认是按值传递的。

例如，使用 VB 语句“Function F1(arg as Integer)”定义的函数，等价于“Function F1(ByRef arg as Integer)”，当函数体内部修改了参数值时，跳出函数后该值也发生了改变。

```
Function F1( arg as Integer )
    arg = 10
end Function
```

此时语句“Function F1(arg as Integer)”等价于“Function F1(ByRef arg as Integer)”，可以执行以下 VB 代码：

```
Dim Val as Integer
Val = 20
F1( Val )
//执行到这里， Val 变量值在函数体内部被修改为 10
```

由于在 VB 中调用方法默认是按引用传递参数的，这容易留下难以察觉的错误，因此写 VB 代码的时候都推荐显式地写出 ByVal 或 ByRef 的参数传递方式。

但是在 C# 中默认是按值传递的，例如对于以下 C# 代码：

```
void F1( int arg )
{
    arg = 10 ;
}
```

若执行以下 C# 代码：

```
int Val = 20;
Fl( Val );
```

那么 Val 变量值还是 20。

在 C# 中，若要按引用传递参数则需要使用关键字“ref”来修改参数，即写成类似“void Fl(ref int arg)”的代码。这样在函数体内部修改参数值，函数体外部便可获得修改后的参数值。

C# 提供“out”关键字为参数的传递方式提供了更精细的控制，标记了“out”的关键字说明该参数按引用传递，而且必须在函数体内部修改该参数值，否则会出现编译错误。这也算是非主流的函数返回值吧。

对于类类型，VB 和 C# 都是按引用传递的。

- 函数调用方式

VB 调用方法时直接写函数名，后面跟着参数，参数不用括号分开。比如定义了函数“Function Fl(byval a as Integer, byval b as Integer)”，则调用该函数的代码通常为“Fl 10, 20”，少数情况也可以写成“Call Fl(10, 20)”。但对于 C# 必须在参数两边写上圆括号，因此必须写成“Fl(10, 20)”。

- 异常处理

VB 中可以使用“on error goto”或“on error resume”来处理异常，并支持使用“resume”语句来跳回到发生异常的代码处。但 C# 不支持类似“on error resume”语句的功能，也不支持“resume”进行代码跳转。

6.10.2 从 C/C++ 到 C#

笔者在此提出一些从 C/C++ 语言转变到 C# 语言的一些参考建议，希望能对那些已经掌握了 C/C++ 语言的读者提供一些帮助。其要点如下。

- 指针

C 语言最强大的功能就是指针。C# 也支持指针，但已经不推荐使用，从而使指针成为很高级的 C# 语法了。

在 C# 语言中要使用指针，首先要使用语法结构“unsafe{ }”创建一个代码块，在这个代码块中才能使用指针。

定义指针变量时，C# 中“*”必须和基础类型一起使用，否则会出现编译错误，比如代码“int* p1, p2, p3”在 C# 中表示定义了 3 个指向整数类型的指针变量；代码“int *p1, *p2, *p3;”在 C# 中是会出现编译错误的，而在 C 语言中是正确的。

C# 不允许不同指针类型之间或指针类型与整数之间的转换，而 C 语言是允许的。比如以下代码：

```
int* p1;
```

```
char*p2;  
p1 = p2;
```

这段代码在 C# 中是错误的，而在 C 语言中是可以的。

- 安全性

C# 提供了比 C/C++ 更为严格的安全性。比如 C/C++ 不检查数组下标是否越界，而 C# 是会严格检查的。 .NET 框架提供了可信运行环境，在此上面运行的所有程序都要受到限制，包括 C#。

- 条件编译

C# 支持类似 C 语言的条件编译功能，支持 “#define”、“#undefine”、“#if” 等指令。但不支持 C 样式的宏，包括宏常量和带参数的宏，也不支持 “#include” 指令。

- 访问类型成员

在 C++ 中使用 “->” 访问类型成员，而在 C# 中直接使用一个字符点 “.” 来访问类型成员。

- 函数

在 C/C++ 中，函数是可以独立编写的，不属于任何类型，比如 Main 函数。但在 C# 中没有游离的函数，所有的函数必须属于某个类型，包括 Main 函数。

- 字符串定义

在 C/C++ 中使用双引号定义一个字符串量，并且使用斜杠字符 “\” 进行转义；C# 也支持这种方式，不过还提供一种定义多行字符串的方法，也就是在双引号前面加上 “@” 符号。例如以下 C# 代码定义了一个多行字符串。

```
string myStr = @"abc  
1234 \t  
5678";
```

在 C# 定义的多行字符串中，斜杠转义是无效的。

要特别注意，在传统的 C/C++ 中，字符串是以 “\0” 作为字符串的结束标识的，而在 C# 中字符串对象是可以包含 “\0” 的，这点在两者之间传递字符串数据时要注意。

- 继承

在 C++ 中，一个类可以继承多个其他类，可以实现多个接口；而在 C# 中一个类只能继承一个其他的类，也可以实现多个接口。

- 注释

C/C++ 支持使用双斜杠字符 “//” 定义单行注释，使用斜杠和星号对 “/* */” 实现多行注释，C# 也支持这种方式。

不过 C# 更进一步，在类型或类型成员的定义前面，使用连续的三个斜杠 “///” 和特定的 XML 语法实现文档化的注释，C# 编译器能提取出代码中的文档化的注释并自动生成相关的 SDK 帮助文档。

6.10.3 从 Java 到 C#

笔者以个人对 Java 语法的一点理解来提出从 Java 转到 C#的建议，希望能对已经掌握了 Java 语法的读者有所帮助。

- Java 和 C#语法确实非常像，代码组织结构也很类似，双方都有相对应的关键字。
- C#支持委托，而 Java 不支持。C#中的委托可以说是 C 语言的函数指针的 OOP 版本，它是一种变量，能存储某个函数的入口点地址，因此委托可以增强 C#语言的灵活性，而 Java 不支持这种特性。
- C#支持事件，而 Java 不支持。C#在委托的支持下支持事件，一个事件上可以绑定多个委托来支持广播；而 Java 需要通过实现接口的方式来模拟实现事件机制，具体操作不太方便。

第 7 章

第一次 C# 体验

在学习了 C# 的基本语法后，在本章带领读者开始对 C# 世界的第一次探索，进行第一次 C# 体验。

7.1 第一次使用 VS.NET 集成开发环境

笔者从未见过有人完全用记事本程序编写 C# 代码来开发程序，而全部都是使用某种开发工具来开发 C# 程序的。在各种开发工具中，最常用的是微软官方提供的 VS.NET 集成开发环境。本节笔者就带领读者第一次使用 VS.NET 集成开发环境。

VS.NET 集成开发环境 (Microsoft Visual Studio) 是微软推出的应用系统开发工具套件。下面以 VS.NET 2010 旗舰版为例说明 VS.NET 的使用。

安装 VS.NET 2010 后，启动 VS.NET，打开一个工程，即可看到其主界面，如图 7-1 所示。

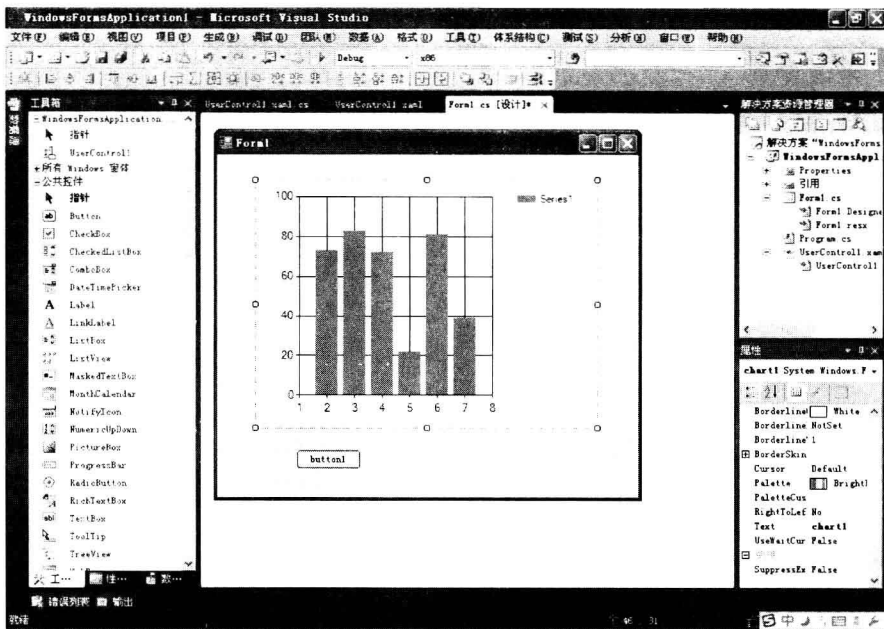


图 7-1 IDE 主界面

这是一个比较复杂的用户界面，大体可以分为 6 个部分。

7.1.1 菜单栏

VS.NET 的菜单栏包含了很多菜单项目，通过它能调用 VS.NET 的大部分功能。随着主工作区的内容不同，菜单栏的内容也可能有所改变。

7.1.2 工具条

工具条上的功能按钮非常多，列出了最常见的功能。工具条中的内容可能随着主工作区的内容不同而发生改变。

7.1.3 工具箱

IDE 主界面的左边是工具箱，最重要的功能就是列出了可用的控件。用户双击工具箱中的控件项目或将控件项目拖曳到窗体上，就能在窗体上新增一个控件。

7.1.4 主工作区

IDE 主界面的中间是主工作区，主工作区以分页标签的方式显示了若干个文档，各个文档类型可能不一样。图 7-1 中显示的就是一个 WinForm 窗体文档，用户可以在其上面新增、修改和删除控件，使用鼠标拖曳操作来移动控件的位置或改变大小。

7.1.5 解决方案资源管理器

IDE 的右上半部分是解决方案资源管理器，如图 7-2 所示。这是一个树状列表，根节点就是解决方案节点，第二层节点就是 C#工程，以下是 C#工程下面的各种各样的成员。

在 VS.NET 中，只能打开一个解决方案，解决方案的文件扩展名为“.sln”，即使只打开一个 C#工程文件（扩展名为“.csproj”。），系统也会尝试找到该工程文件所管理的解决方案文件，若没有找到则创建一个新的解决方案。因此该资源管理器的根节点一定是一个解决方案节点。

一个解决方案可以包含若干个工程项目，比如既可以包含一个 C#的 WinForm 项目，也可以包含一个 VB.NET 的 ASP.NET 项目，或者其他项目。

不同类型的工程项目节点，其子节点是不一样的。例如对于 C#工程项目，就具有以下几个

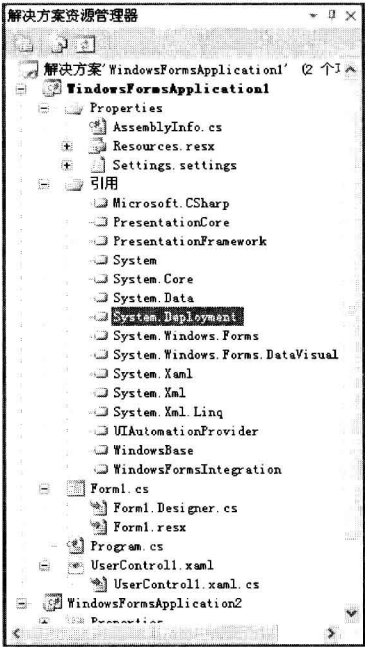


图 7-2 解决方案资源管理器

子节点。

Properties 节点：该节点下面放置了程序版本信息定义文件“AssemblyInfo.cs”、全局的资源文件“Resources.resx”和应用程序配置信息文件“Settings.settings”。

引用节点：该节点下面列出了应用程序所引用的其他.NET 程序组件。程序要编译、运行和部署，就必须保证计算机中存在这些被引用的程序集。

文档节点：工程节点下面有若干个文档节点，这些节点可以是窗体文件、C#源代码文件、资源文件或其他文档。

子目录：工程节点下面可以包含子目录结构，这样以多层的目录结构来安排各种文档，便于管理。比如可以将属于某个程序模块的所有源代码文件放置在一个单独的子目录下，子目录名称就是程序模块的名称。

7.1.6 属性编辑器

IDE 的右下半部分是属性编辑器。该属性编辑器中列出了用户当前选中的对象的属性。这个属性编辑器是一个很重要的应用程序设计工具，方寸之间保罗万象。例如，在窗体设计器中选一个按钮控件，则会出现属性编辑器的用户界面，如图 7-3 所示。



图 7-3 属性编辑器

等。

(2) 字母排序：按下该按钮，属性列表中的内容不进行分类，统一按照属性名的字母顺序排序。

(3) 属性：按下该按钮，属性列表中显示的是当前组件的各个属性。

(4) 事件：按下该按钮，属性列表中显示的是当前组件的各个事件。

1. 组件下拉列表

在属性编辑器中，最上面是一个组件下拉列表，它列出了窗体中所有可编辑的组件名称及其类型名称。当窗体中控件数量太多或相互遮盖不易选择时，可以通过这个下拉列表来选中某个组件。

2. 工具条

组件下拉列表下面就是属性编辑器工具条，该工具条有以下按钮。

(1) 按分类排序：按下该按钮，属性列表中的内容就按照分类进行排序，同属一个分类的属性排在一起。一般可分为“布局”、“行为”、“设计”、“数据”、“外观”、“格式”

3. 属性名称列表

属性编辑器的左半边是属性名称列表，列出了当前对象的所有可编辑的属性的名称。有些属性名称前面有一个小加号，可以展开这个属性项目从而查看和编辑它的子属性值。

4. 属性值列表

属性编辑器的右半边是属性值列表，显示了所有属性的属性值，用户可以在属性值列表中直接修改属性值。同时系统会以丰富多彩的方式显示属性值，并提供多种方式编辑属性值。

粗体：组件在设计时可以为属性设置默认值，若当前属性值等于默认值，则属性值将以正常字体显示，否则以粗体显示。比如对于 BackColor 属性，其默认值为“Control”，而此处设置为“Red”，不等于默认值，因此以粗体显示属性值“Red”。

自定义绘制数值：有些属性值不只是简单地显示文本，还会在属性值前面的一个小方框内绘制表示数据的图形。例如属性编辑器中的 BackColor 属性，当前值为“Red”，则颜色名称前面会显示一个小红框来显示该颜色值。

扩展编辑器：一般的用户可以直接在属性值文本框中输入属性值。不过属性编辑器支持扩展属性编辑器。比如选中 BackColor 属性，在其值后面出现一个小的下拉按钮。若单击此下拉按钮，则会显示如图 7-4 所示的颜色值选择列表。

用户可以使用该列表选择合适的颜色值，用户双击所需的项目或单击列表之外的地方，该列表会关闭并根据用户的选择来修改属性值。

属性编辑器也提供对话框的方式来扩展编辑属性值。例如选中 Font 属性，则该属性项目如图 7-5 所示。



图 7-4 颜色值选择列表

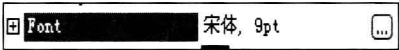


图 7-5 属性项目

此时属性值后面出现一个小按钮，单击该按钮会弹出如图 7-6 所示的对话框。

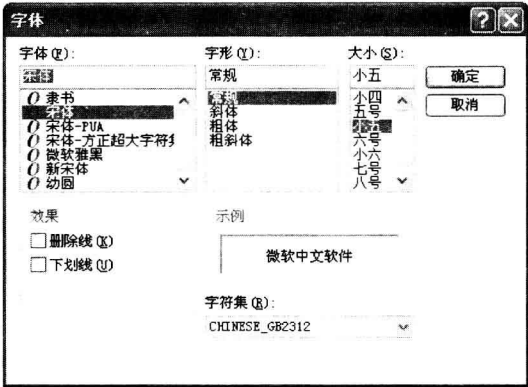


图 7-6 “字体”对话框

通过该对话框就能很方便地设置属性值了。

5. 控件设计动作区

有些控件支持某些设计时的动作，当选中这类控件时，属性编辑器中会出现可选设计动作面板。比如选中一个分页标签控件，则属性编辑器中会显示一个分页标签可选设计动作面板。用户可以单击面板中的标签来执行一些常见的设计动作，如图 7-7 所示。



图 7-7 面板

6. 属性说明区

属性编辑器最下面是当前属性说明区域，显示了当前属性的相关说明。比如当选中 BackColor 属性时，此处显示“组件的背景色”。

7.2 C#程序类型

了解了 VS.NET 的主界面后，现在介绍 VS.NET 所能开发的 C#的程序类型。

在图 7-8 中单击 VS.NET 开发环境中的菜单项目“文件→新建→项目”，则可显示如图 7-9 所示的“新建项目”对话框。

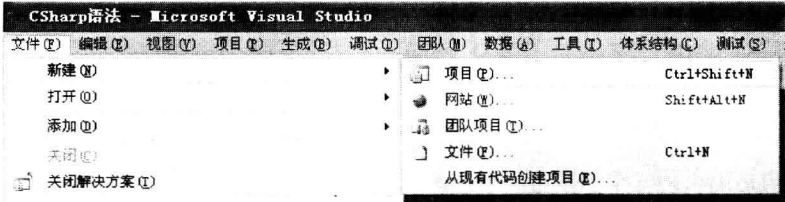


图 7-8 单击菜单项目

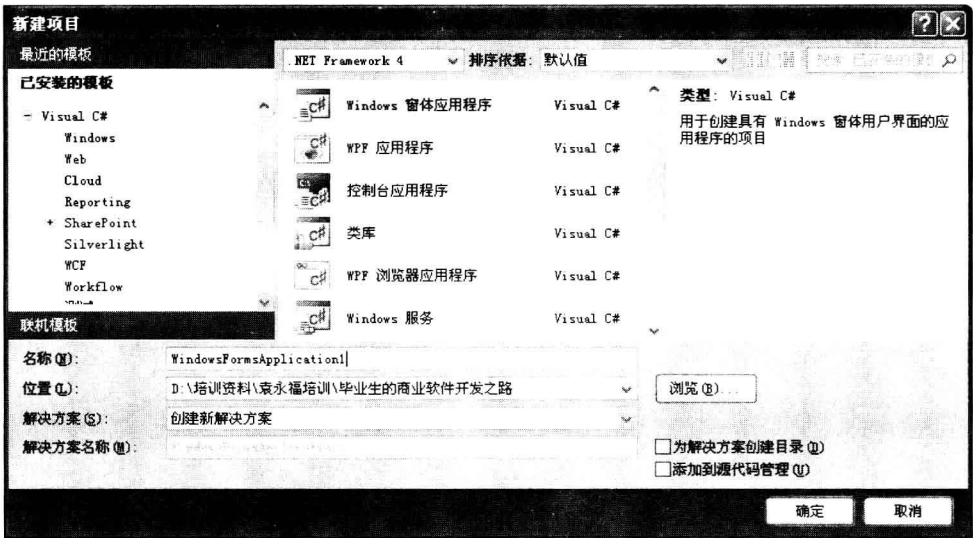


图 7-9 “新建项目”对话框

可以看到使用 C# 可以开发很多类型的程序。对于初学者来说，主要有：Windows 应用程序、ASP.NET Web 应用程序、命令行应用程序和其他类型的应用程序。

7.2.1 Windows 应用程序

Windows 应用程序是最常见的程序类型，其程序文件的扩展名为 EXE，比如“Hellow.exe”。Windows 应用程序一般都具有图形化用户界面，也就是会显示一个窗体，窗体上有菜单、按钮之类的控件，用户操作用户界面上的控件即可完成某项功能。

VS.NET 可以为移动设备开发应用程序，比如智能手机。从广义上来说，移动设备用的程序也算是 Windows 应用程序，只不过这些程序运行在移动版的 Windows 中。

7.2.2 ASP.NET Web 应用程序

ASP.NET Web 应用程序就是完全用 Web 浏览器作为用户界面的 B/S 程序。在 VS.NET 中，这种程序的用户界面设计方式和 Windows 应用程序的有些类似。微软在制作 VS.NET 集成开发环境时一直试图减少 Windows 应用程序和 ASP.NET Web 应用程序两者开发过程的差别。

此外，Web Service 程序也是 ASP.NET Web 应用程序。

7.2.3 命令行应用程序

命令行应用程序就是字符用户界面的程序，它一般没有复杂的用户界面，仅仅通过命令行的方式与使用者交互。命令行应用程序应用不多，但它支持命令行参数，适合专业人士快速调用。因为很多时候，专业人士用键盘输入命令执行操作的方式比图形用户界面上的鼠标操作要快捷。

7.2.4 其他类型的应用程序

当然使用 C# 不仅能开发 Windows 应用程序和 ASP.NET Web 应用程序，还能开发很多种其他类型的程序，比如 Windows 服务程序等。

开发第一个 Windows 应用程序

在本章使用 VS.NET 2010 开发第一个 Windows 应用程序，可初步了解如何使用 C#进行 Windows 应用开发。

本软件的功能需求就是开发一个能进行加、减、乘、除的计算器。

8.1 建立 C# Windows 应用程序项目

单击 VS.NET 开发环境中的菜单项目“文件→新建→项目”，出现如图 8-1 所示的“新建项目”对话框。

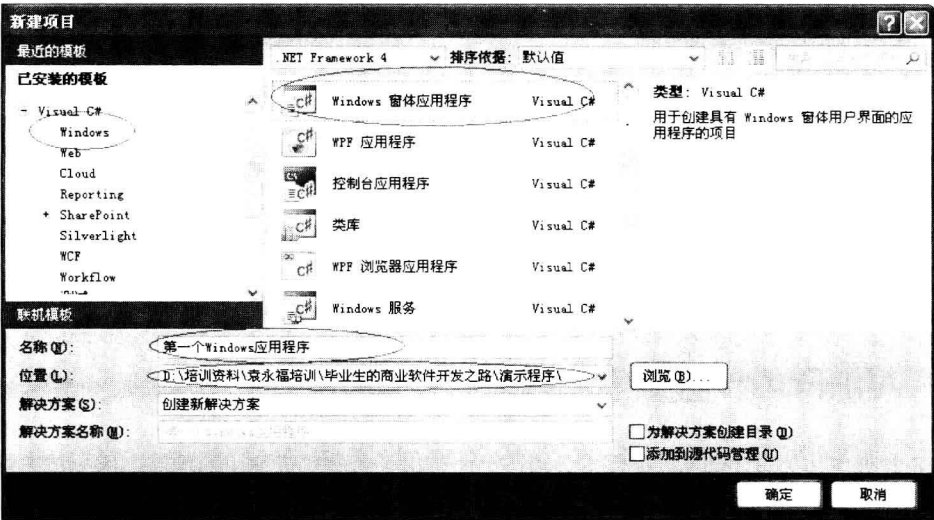


图 8-1 “新建项目”对话框

在该对话框中，进行以下四步操作。

- (1) 在“已安装的模板”树状列表中选中“Windows”节点。
- (2) 在右边的列表中选中“Windows 窗体应用程序”项目。
- (3) 在“名称”文本框中输入“第一个 Windows 应用程序”。

(4) 在“位置”文本框中输入保存程序文件的目录路径。

单击“确定”按钮关闭该对话框，VS.NET 会自动生成一个 Windows 应用程序的框架。此时 VS.NET 开发环境界面如图 8-2 所示。

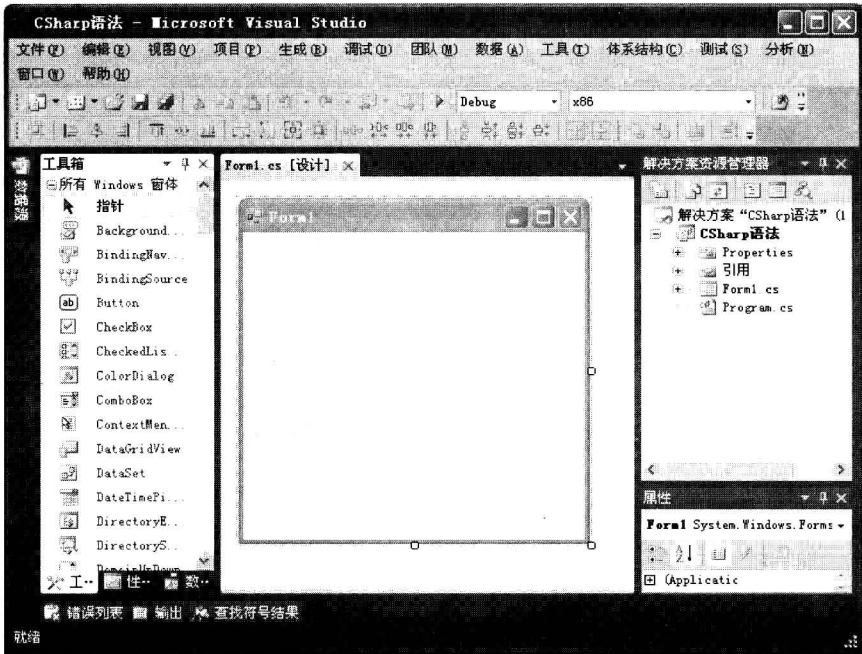



图 8-2 VS.NET 开发环境

学过 VB 的可以发现，这个用户界面和 VB 的用户界面很相似。对于会 VB 的 C#初学者来说，使用 C#开发 Windows 应用程序和 VB 差不多，只是后台编程语言由 VB 改成 C#。

在 VS.NET 的用户界面中，最左边是控件工具箱，中间是窗体设计器，右上方为工程资源树状列表，右下方为控件属性编辑列表。

8.2 WinForm 控件工具箱

图 8-2 最左边为 Windows 控件工具箱，上面列出了所有可以使用的控件。用户按下该列表中的某个控件图标，然后将鼠标拖曳到左边的窗体上即可在窗体上新建一个指定类型的控件。如图 8-3 所示，用户按下“Button”项目然后拖曳到窗体上，会在窗体上新增一个按钮控件，控件的位置就是鼠标光标所在的位置，控件大小采用默认大小。

当工具箱中的控件项目按下后，将鼠标移动到窗体设计器中，此时鼠标光标就变成附加控件图标的十字形。若用户按下“Button”项目，然后将鼠标移动到窗体设计器中，则鼠标光标样式为“”。此时开发人员在窗体设计器中，使用鼠标拖曳操作可在窗体上画出一个指定位置、指

定大小的控件。

在默认情况下，控件工具箱中列出的控件是够用的。但这里并没有列出微软.NET 框架自带的所有 Windows 控件，而且也不包含第三方的控件，为了使用没有默认列出的控件，需要向控件工具箱上添加新的控件。

如图 8-4 所示，用鼠标右击工具箱，在弹出的快捷菜单中单击“选择项”，会显示如图 8-5 所示的“选择工具箱项”对话框。

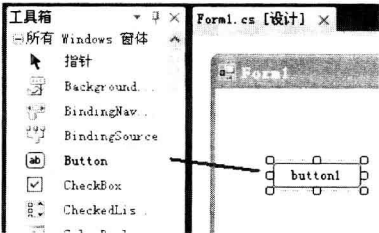


图 8-3 Windows 控件工具箱

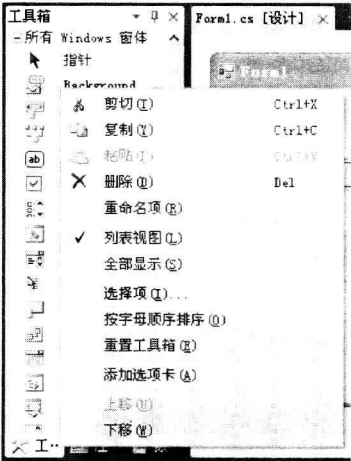


图 8-4 快捷菜单

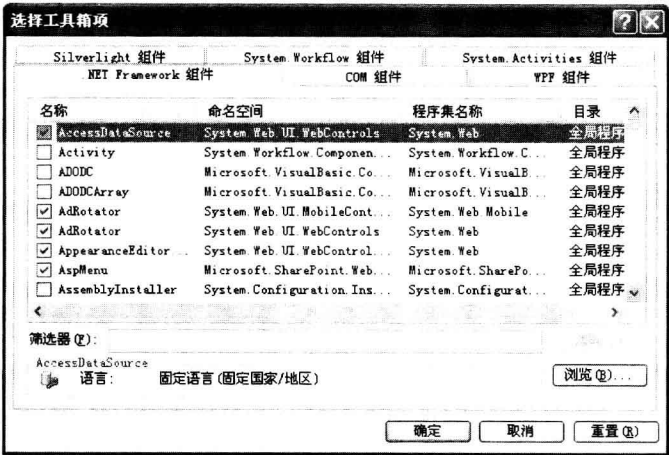


图 8-5 “选择工具箱项”对话框

在该对话框中最重要的是“.NET Framework 组件”选项卡。该选项卡列出了本系统安装的基于.NET 平台的组件。

在图 8-5 中列出了目前 VS.NET 所能识别的所有 Windows 控件，可以在某个控件前面打钩使该控件能显示在工具箱的控件列表中，不打钩就不显示。

对于从第三方获得的已经开发好的控件，开发者可以单击“浏览”按钮来选择包含控件的.NET 程序集文件，比如“MyControls.DLL”。然后控件列表中便出现了开发人员选择的.NET 程序集中所包含的控件项目，开发人员可以在这些控件前面打钩使其显示在工具项目中的控件列表中。

8.3 WinForm 窗体设计器

VS.NET 的 Windows 窗体设计界面的中间是窗体设计器。其用户界面如图 8-6 所示。

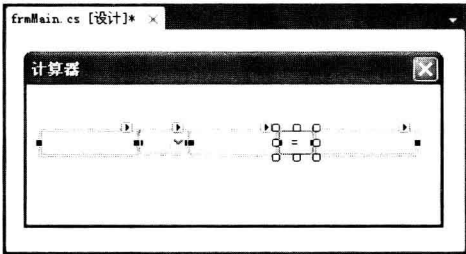


图 8-6 窗体设计器

窗体设计器具有以下几个主要功能：

- (1) 开发人员可以从左边的控件工具箱中拖曳图标到窗体上，以便在窗体中放置各种各样的控件，控件的位置就是鼠标拖曳到的位置，控件的大小为默认大小；开发人员也可以按下控件工具箱中的控件图标，然后在窗体中拖曳，画出指定位置、指定大小的控件。
- (2) 开发人员可以很方便地使用鼠标拖曳操作，来移动控件在窗体中的位置或修改控件的大小。当窗体中存在可容纳子控件的容器控件时，开发人员可以将控件从一个容器控件中拖曳到另外一个容器控件中，这样能修改控件之间的父子关系。
- (3) 开发人员可以单击选中某个控件，然后在右边的属性列表中查看和修改控件的属性。
- (4) 开发人员可以双击控件，而且由于控件具有默认事件，因此用户界面会切换到 C# 代码编辑界面，并自动构造出响应控件默认事件的 C# 代码结构。开发人员可以直接输入响应控件事件的程序代码。

在图 8-6 中，窗体上已经放置了两个 Label 控件，两个 TextBox 控件和两个 Button 控件，并设置了控件的位置、大小和一些属性。当运行程序时，该窗体的用户界面如图 8-7 所示。

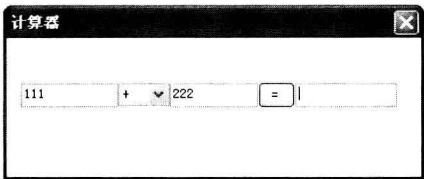


图 8-7 用户界面

可以看到窗体在设计时的显示效果基本上等于运行时的显示效果，实现了一种“所见即所得”的可视化界面设计功能。

1. 可视化软件开发

微软的集成开发环境“Microsoft Visual Studio”中的 Visual 是“可视化”的意思。微软最早在 Visual Basic 中实现了可视化的用户界面设计功能，这引起了软件开发方法的革命。

在没有可视化程序开发时，开发人员需要手动编写以下代码来构造出图 8-7 所示的用户界面。

```
private System.ComponentModel.IContainer components = null;

private System.Windows.Forms.TextBox txtNumber1;
private System.Windows.Forms.ComboBox cboOperator;
private System.Windows.Forms.TextBox txtNumber2;
private System.Windows.Forms.Button btnCalculate;
private System.Windows.Forms.TextBox txtResult;

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

private void InitializeComponent()
{
    this.txtNumber1 = new System.Windows.Forms.TextBox();
    this.cboOperator = new System.Windows.Forms.ComboBox();
    this.txtNumber2 = new System.Windows.Forms.TextBox();
    this.btnCalculate = new System.Windows.Forms.Button();
    this.txtResult = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // txtNumber1
    //
    this.txtNumber1.Location = new System.Drawing.Point(12, 40);
    this.txtNumber1.Name = "txtNumber1";
    this.txtNumber1.Size = new System.Drawing.Size(84, 21);
    this.txtNumber1.TabIndex = 0;
    //
    // cboOperator
    //
    this.cboOperator.DropDownStyle = System.Windows.Forms.ComboBoxStyle.
DropDownList;
    this.cboOperator.FormattingEnabled = true;
    this.cboOperator.Items.AddRange(new object[] {
        "+",
        "-",
        "*",
        "/" });
    this.cboOperator.Location = new System.Drawing.Point(96, 40);
    this.cboOperator.Name = "cboOperator";
    this.cboOperator.Size = new System.Drawing.Size(44, 20);
    this.cboOperator.TabIndex = 1;
```

```
//  
// txtNumber2  
//  
this.txtNumber2.Location = new System.Drawing.Point(140, 40);  
this.txtNumber2.Name = "txtNumber2";  
this.txtNumber2.Size = new System.Drawing.Size(76, 21);  
this.txtNumber2.TabIndex = 0;  
//  
// btnCalculate  
//  
this.btnCalculate.Location = new System.Drawing.Point(216, 39);  
this.btnCalculate.Name = "btnCalculate";  
this.btnCalculate.Size = new System.Drawing.Size(32, 23);  
this.btnCalculate.TabIndex = 2;  
this.btnCalculate.Text = "=";  
this.btnCalculate.UseVisualStyleBackColor = true;  
//  
// txtResult  
//  
this.txtResult.Location = new System.Drawing.Point(248, 40);  
this.txtResult.Name = "txtResult";  
this.txtResult.ReadOnly = true;  
this.txtResult.Size = new System.Drawing.Size(87, 21);  
this.txtResult.TabIndex = 0;  
//  
// frmMain  
//  
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 12F);  
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;  
this.ClientSize = new System.Drawing.Size(356, 122);  
this.Controls.Add(this.btnCalculate);  
this.Controls.Add(this.cboOperator);  
this.Controls.Add(this.txtResult);  
this.Controls.Add(this.txtNumber2);  
this.Controls.Add(this.txtNumber1);  
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;  
this.MaximizeBox = false;  
this.MinimizeBox = false;  
this.Name = "frmMain";  
this.Text = "计算器";  
this.ResumeLayout(false);  
this.PerformLayout();  
}
```

让开发人员编写这样的代码是枯燥乏味且效率低下的，人们在编写这种代码的过程中需要在脑海中想象出这段代码运行后的窗体界面。需要多次运行程序显示窗体并查看运行效果后修改代码，然后再进行控件位置、大小的微调。当窗体界面中需要新增或删除控件时会影响到其他控件的排版，此时又要多次运行程序显示窗体来修改代码并进行微调。

当窗体界面很复杂时，开发人员需要编写和维护几千行的构造窗体用户界面的代码，这一大段代码开发和维护成本是非常高的。

而使用可视化的窗体用户界面设计功能能大大降低设计和维护窗体用户界面的成本。在窗体设计器中，开发人员可以看到控件在窗体中的排版效果，能比较精确地进行控件排版微调，还可以使用控件的属性编辑器来编辑控件属性，并能实时地看到用户界面的修改效果。这样能大幅提

高用户界面设计的工作效率，缩短开发和维护时间。

从开发方式来说，可视化用户界面设计降低了劳动强度。在没有可视化设计前，开发人员全靠写代码来拼凑软件用户界面。写代码是一种高强度的劳动，开发人员需要端坐在计算机前，从脑子里想出代码，然后双手在键盘上快速地输入代码，此时无论身体还是精神都处于紧张状态，劳动强度高，比较容易疲劳。

在可视化用户界面设计过程中，大部分操作是靠鼠标来完成的，单手即可完成。因此开发人员不需要端坐，可以采用比较休闲、舒适的坐姿，而且设计过程比较简单，大脑思考不多，此时身体和精神处于较为放松的状态，劳动强度低，不容易疲劳。

有人认为可视化设计降低了对软件开发人员的要求，对于其成长不利。其实并不是这样的。

可视化操作掩盖了软件用户界面的后台实现，由于人们普遍具有惰性，使用可视化操作完成用户界面的设计任务后就不深究其后台的实现，这确实不利于开发人员的技术钻研。不过学海无涯，软件开发涉及很多的技术细节，若让开发人员了解软件开发过程中的所有技术细节，尤其是构造用户界面的比较枯燥乏味的技术细节，这会耗掉他们相当多的精力，从而减少了对其他技术细节的关注。

现代软件的开发内容分为技术内容和业务内容两种。对于行业应用软件，开发人员应将主要精力放在行业业务细节上，需要用很多精力来理解客户业务需求，包括客户的业务数据和执行流程。只有深入了解了业务，开发人员才能利用软件开发技术来开发符合业务内容的应用软件。因此开发人员应把主要精力放在深入了解客户业务上，充分掌握客户的业务细节。

由于人的精力是有限的，因此开发人员应当尽量降低对技术细节的精力消耗，这就是各种应用系统框架出现的原始需求。可视化程序设计能隐藏大量的技术细节，使开发人员能快速高效地设计程序，降低花在软件开发技术细节上的精力消耗。这优化了开发人员的精力配置，使开发人员不必纠缠在软件技术细节问题上，更侧重于对业务的理解和实现，提高软件开发的生产效率。

另外可视化软件开发操作简单，不需要手动写代码，这能降低对软件开发人员的技术要求，使大量的低级软件开发人员也能参与商业软件的开发。

一般来说，一个企业的软件开发人员中高水平的开发人员少，低水平的开发人员多。高水平的开发人员能开发和维护上万行的源代码，低水平的开发人员只能维护几千行的源代码，而高校实习生只能编写几百行的源代码。因此不能指望低水平的开发人员通过编写程序代码来构造软件的用户界面。

若没有可视化软件开发，则企业中必然存在大量的不能投入实际工作的开发人员，这对企业来说会造成巨大的不必要的成本浪费。而可视化软件开发掩盖了一些技术细节，使低水平的开发人员能比较独立地进行软件的用户界面开发，这样企业中所有的开发人员都能投入实际工作，不会浪费人力资源，符合企业的利益。

当然，完全使用可视化软件设计确实纵容了开发人员的惰性，大量的没有钻研精神的软件开发人员只利用现成的可视化软件设计的功能，而不去思考其掩盖的技术细节，这不利于低水平的

软件开发人员升级为高水平的软件开发人员。笔者见过这类的开发人员，使用 VS.NET 从事开发工作若干年，只会通过拖曳生成 DataSet，然后在窗体上放置若干个控件，直接绑定到数据源上，几年下来都在干这样的工作。对此，少数开发人员觉得工作一段时期，自己的水平没有多大提高而有所焦虑并试图改变；当然也有很多懒惰的开发人员对此麻木不仁，技术水平原地踏步，终难成大器。

因此有抱负的开发人员还需要大力发扬钻研精神，不拘泥于可视化软件设计带来的便利，还需要深入理解相关技术细节，这是初级开发人员成长的必由之路。

有完美主义者认为让初学者学习软件开发，必须抛弃可视化软件设计，用手工编写代码来实现一切功能，这样做是想让学习者一开始就学习技术底层细节，笔者觉得不妥。

从软件开发学习者的角度来说，一开始就学习技术底层细节就像让小学生学习高难度的奥数一样，对于少数天才这是提升能力的较好方式，但对于大多数普通人来说是拔苗助长。过高的难度将打击学习者的信心，使学习者丧失对软件开发的兴趣，容易导致其放弃学习。

持精英教育观点的人士可能会说只有这样才能大浪淘沙，才能发现并培养出高水平的软件开发人员。但是中国软件行业需要几十万上百万的软件开发人员，若通过这种方式培养真不知何年何月才能凑齐这个数字。

从企业的角度来说，一开始就让初学者或毕业生学习繁杂的技术底层细节，可能需要好几个月甚至一年以上的的时间，还得让老员工花时间悉心指导才可能成功；若经过培训发现初学者不适合做软件开发工作，那么之前的工作全部浪费。这种培养方式对自负盈亏的企业来说，成本太高了，得不偿失。

若让初学者首先学习可视化软件设计，则学习难度低，速度快，而且能较快地投入实际软件开发过程中。因此应让初学者或大学毕业生尽快投入实际工作，为企业带来工作业绩；而且毕业生看到自己能投入实际工作，也是一种激励，使其能较快地建立起职场自信心，产生对软件开发工作的兴趣，这对企业和个人来说都有好处。

综上所述可以看到，可视化软件设计确实较大地提高了开发人员的工作效率，降低了软件的开发成本，也有利于毕业生初期适应软件开发工作。

8.4 Windows 窗体设计概念及原则

本节介绍 Windows 窗体设计中的一些概念和原则。

1. 控件的名称

当开发人员在窗体中新增一个控件时，系统会给这个控件设置默认名称。比如对于 Label 控件，其默认名称可能为 label1、label2 等，对于 TextBox 控件其默认名称可能为 textBox1、textBox2 等。

如图 8-8 所示为“用户信息”窗体，图中显示了各个控件默认的名称。

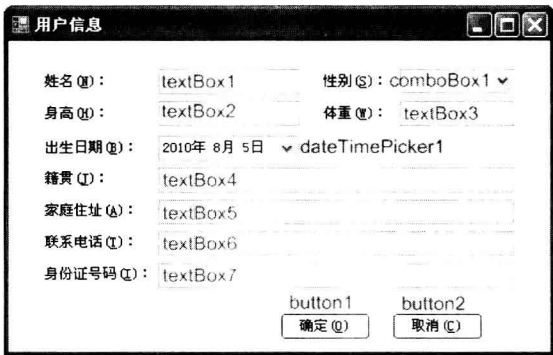


图 8-8 各个控件默认的名称

这是一种非常糟糕的窗体设计结果。在编写该窗体的 C#代码时，由于无法同时看到窗体设计界面，因此很难记下各种控件的名称及其功能。当窗体控件比较多时，程序代码的编写和维护都相当困难。

为了避免这种不好的情况，开发人员应当养成在窗体上新建控件就立即修改控件名称的习惯，这样能有效地改进软件的用户界面的设计质量，也是提高窗体后台代码质量的基础。

不过对于 Label 等少数几种控件，由于基本上不参与编程，只是在用户界面上显示一段不可改变的静态文本，因此可以不设置 Label 控件的名称。但在少数情况下，当窗体后台代码需要操作 Label 控件的文本等属性值时，为了提高代码的质量也应当修改 Label 控件的名称。

对各个数据输入控件的名称设置如图 8-9 所示。

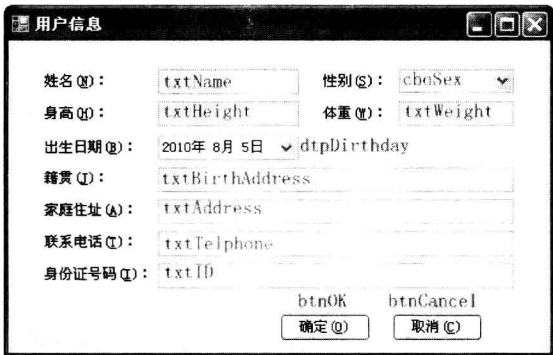


图 8-9 各个数据输入控件的名称设置

与代码变量命名规则类似，窗体控件也有命名规则，业界主流的控件命名规则有多种，各种规则之间差别不大，一般采用的命名规则大致如下：

控件名称一般由表示控件类型的前缀加上表示控件作用的字符组成。常用控件类型和前缀的对应关系如表 8-1 所示。

表 8-1 常用控件类型和前缀的对应关系

控 件 类 型	前 缀
Button 按钮	btn
CheckBox 复选框	chk
ColumnHeader 视图列表头	col
ComboBox 组合框	cbo
ContextMenu 快捷菜单	ctm
DataGrid 数据网格控件	dg
DataGridView 数据网格视图控件	dgv
DateTimePicker 时间输入框	dtp
DomainUpDown 数值框	dud
Form 窗体	frm
GroupBox 组合框	grp
HScrollBar 水平滚动条	hsb
ImageList 图标列表	img
Label 文本标签	lbl
LinkLabel 带链接的文本标签	lbl
ListBox 列表框	lst
ListView 视图列表	lvw
Menu 菜单	menu
MenuItem 菜单项	menu
NumericUpDown 数值框	nud
Panel 面板	pnl
PictureBox 图片框	pic
ProgressBar 进度条	prg
RadioButton 单选框按钮	rdo
Splitter 拆分条	spl
StatusBar 状态栏	stu
StatusBarPanel 状态栏区域	pnl
StatusStrip 状态栏	stu
TabControl 分页控件	tab
TabPage 选项卡	page
TextBox 文本框	txt
Timer 定时器	tmr
ToolBar 工具条	tbr
ToolStrip 工具栏	tsp
ToolStripButton 工具栏按钮	btn
ToolStripComboBox 工具栏下拉组合框	cbo
ToolStripDropDownButton 工具栏下拉列表	btn

续表

控 件 类 型	前 缀
ToolStripDropDownMenu 工具栏菜单项目	menu
ToolStripLabel 工具栏静态文本	lbl
ToolStripProgressBar 工具栏进度条	prg
ToolStripTextBox 工具栏文本框	txt
TreeView 树状视图列表	tvw
VScrollBar 垂直滚动条	vsb
WebBrowser 浏览器控件	wb

对于其他不常用的控件类型的前缀读者可以自己琢磨或请教他人，控件名称前缀只是一个比较小的细节问题，即使有误也无伤大雅。

一般使用英文或缩写来表示控件的作用，若没有合适的英文则采用汉语全拼音或干脆用中文汉字，但绝不能用中文汉语拼音首字母。因为汉语拼音首字母很难仅仅从字母上猜出其表示的汉字，而猜测英文缩写相对容易得多。

如图 8-10 所示，可能有些人想用汉语拼音的首字母来设置控件的名称，比如将“姓名”文本框取名为“txtXM”，这是一种不好的控件命名方式。因为很多时候很难从拼音首字母中猜测出控件的作用，控件名称的低可读性降低了程序代码的可读性。

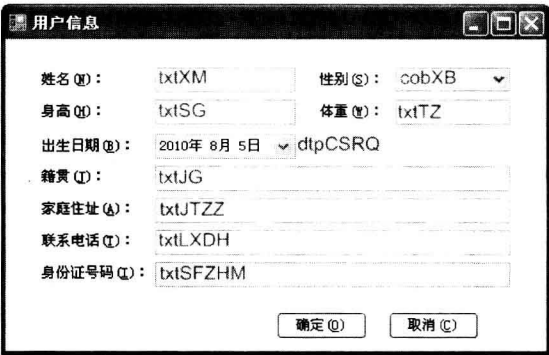


图 8-10 用汉语拼音的首字母来设置控件的名称

因此按照比较合理的命名方式，“姓名”文本框的名称应由 TextBox 控件的前缀“txt”和表示姓名的英文“Name”组合而成，即“txtName”。

在实践中，很多商业软件不是“一次开发，长期使用”的，而是“一次开发，长期修改，长期使用”的。因此开发商业软件需要考虑到未来的修改和升级换代，此时程序代码不但要耐用，还要耐看、耐改，程序代码需要具备良好的可读性，而给控件设置合适的名称就是一项基础工作。其实除了控件的名称，其他诸如数据库表名字段名，窗体名称，程序模块名称等都需要遵守一定的命名规范。

2. TabIndex 属性

设计 Windows 窗体还需要设置好控件的 TabIndex 属性值。在程序运行显示窗体时，用户可以按下 Tab 键来切换输入焦点，而切换顺序可依照各个控件的 TabIndex 属性值从小到大排序。一般输入焦点的切换顺序应该从左到右，从上到下。因此需要设置左上方位的控件的 TabIndex 属性值为小，右下方位的控件的 TabIndex 属性值为大。但实际开发中仍按照具体功能需求来安排输入焦点的切换顺序。

在设置各个控件的 TabIndex 属性值时不应设置为连续的值，比如设置第一个控件的 TabIndex 属性值为 1，第二个控件为 2，第三个控件为 3，等等。采用这种设置时，若未来在输入焦点顺序中插入新的控件，则需要修改大量的控件的 TabIndex 属性值。

因此建议输入的各个控件的 TabIndex 属性值不连续，比如有 10 的间隔，此时可以设置第一个控件的 TabIndex 属性值为 10，第二个控件为 20，第三个控件为 30，等等。这样就为未来插入新的控件留下 TabIndex 顺序的空间，如图 8-11 所示。

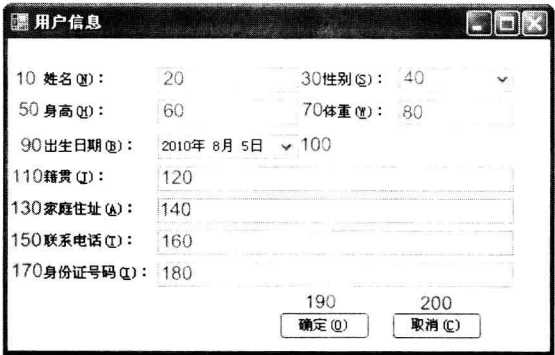


图 8-11 设置不连续的 TabIndex 属性值

在这个“用户信息”窗体设计的例子中，当该窗体运行显示后，用户可以按下 Tab 键从左到右从上到下地切换焦点。

有些控件并不能接受输入焦点，比如 Label 控件，但仍然需要为这类控件设置合适的 TabIndex 属性值。因为 Label 控件虽然不能接受输入焦点，但仍然能接受快捷键。比如在“用户信息”的窗体设计中，设置第一个 Label 控件的文本为“姓名 (&N):”，在程序运行后显示该窗体时，若用户按下“Alt+N”组合键，Label 控件就会感应到快捷键并试图设置输入焦点到自己。由于 Label 控件不能接受输入焦点，因此系统会将输入焦点自动切换到 Label 控件的下一个焦点顺序的控件上，也就是 TabIndex 值为 20 的文本框上。这样用户使用快捷键“Alt+U”就能快速切换焦点到“姓名”文本框了。

当用户界面上数据输入域很多时，这种快速切换输入焦点的功能是很人性化的，方便用户使用纯键盘操作来高速输入和修改数据，显得用户界面做得很专业。若没有这种功能，则用户需要频繁地切换鼠标和键盘来输入数据，大大降低了操作速度。

商业程序的用户界面要求比较高，因为使用者可能好几年天天面对相同的用户界面，累计下来会重复进行几万次操作，此时用户界面需要耐看、耐用。这需要在很多细节上下功夫，其中设置各个控件的 `TabIndex` 属性值就是一项基础工作，必须花点心思做好。

3. Z-Index

在 WinForm 窗体设计中有一个 `Z-Index` 的概念。窗体布局是采用绝对坐标方式的，每一个控件都有 `Left` 和 `Right` 属性，用于确定控件的左上角在窗体中的 `X-Y` 坐标。

当窗体有多个控件时，控件之间可能存在相互重叠的现象，这类似图像处理中层的概念。也就是说，一个控件单独地占有一个控件布局层，多个控件就会有多个布局层的叠加，而越靠近窗体的控件就越容易被覆盖，于是从窗体开始计算，每个控件布局层都有一个从 0 开始计算的序号，这个序号就是 `Z-Index`。

从另外一种方式理解，窗体上的控件布局不仅仅是 `X-Y` 坐标系那种二维的，而是三维的，第三维就是 `Z` 坐标轴，它是从窗体向用户延伸的，控件分布在窗体和用户之间。此时用户是俯看用户界面的，若不注意不会感觉到第三维的存在。控件在第三维中的坐标值就是 `Z-Index`，这个值是从 0 开始的，不会重复；`Z-Index` 值大的控件可以覆盖值小的控件。

还可以从第三种方式看 `Z-Index` 值，窗体对象有一个 `Controls` 的属性，该属性值可以看做一个控件数组，窗体上所有的控件都在这个数组中。系统创建窗体时，是从这个数组逆向遍历获得控件对象，然后依次放置在窗体上的，很显然在数组中靠前的控件可以覆盖靠后的控件，于是控件在这个数组中的逆向序号也可以看做 `Z-Index` 值。

实际上其他软件开发技术中也有 `Z-Index` 属性，比如 VB，还有 Web 开发中的 CSS 样式也支持 `Z-Index` 属性值。

`Z-Index` 与其他控件相关，因此控件本身不会有 `Z-Index` 属性，而是根据控件在窗体中的布局动态地计算出来的。

在 VS.NET 的窗体设计器中，不能直接设置控件的 `Z-Index` 值，但可以使用布局工具条上的“置于顶层”和“置于底层”按钮来将控件沿着 `Z` 坐标轴置于最大值或最小值，也就是沿着 `Z` 坐标轴移动控件，使其最靠近用户或最靠近窗体。

在窗体设计中，设置控件在 `Z` 轴上的先后顺序也是有一些原则的，那就是重要的控件需要置前，不能被不重要的控件所覆盖，如图 8-12 所示。

文本标签的 `Z-Index` 值大于文本框的，这种用户界面是很不好的，会遮挡文本框中的部分类型，而且也不美观。因此需要调整为文本框靠前，也就是达到如图 8-13 所示的效果。



图 8-12 调整前



图 8-13 调整后

在排版时如果用户界面被迫发生一些重叠遮盖，则必须保证重要的控件不被遮盖。

一般来说，若设计人员心中已经知道用户界面的设计，在窗体上从左到右、从上到下地依次添加控件，控件的 TabIndex 和 Z-Index 值都是不断增加的，此时做好后一般无须调整。但在实际中设计好的窗体经常会添加或删除控件，或者大幅移动控件的位置，此时需要注意调整控件的 TabIndex 和 Z-Index 值。

4. 统一的控件对齐方式

在同一个软件中，甚至同一家企业开发的所有软件中，对窗体控件纵向应当采用相同的对齐方式，可以左对齐、居中对齐或右对齐。在此建议左对齐。

对于“用户信息”窗体，控件设计中采用左对齐时窗体运行时的用户界面如图 8-14 所示。

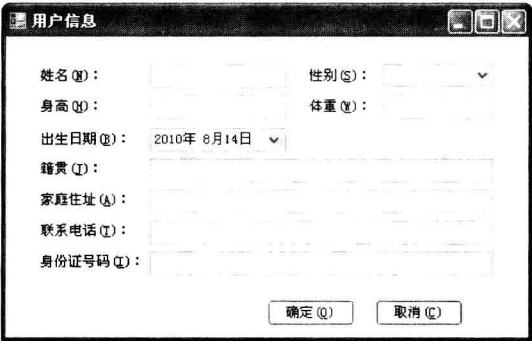


图 8-14 用户界面（左对齐）

采用居中对齐时窗体运行时的用户界面如图 8-15 所示。

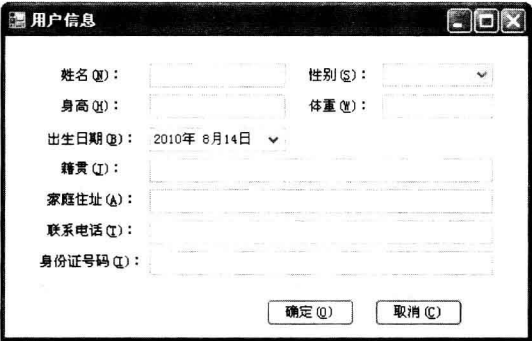


图 8-15 用户界面（居中对齐）

采用右对齐时窗体运行时的用户界面如图 8-16 所示。

对比这三种情况，应该是控件左对齐排版最好，其次是右对齐，居中对齐最差。因此一般都采用左对齐方式。

不管采用哪种控件对齐方式，一定要记住在同一个软件中所有用户界面都应采用统一的对齐方式，不能一个窗体采用左对齐而另一个窗体采用右对齐，这是一种相当不专业的表现。

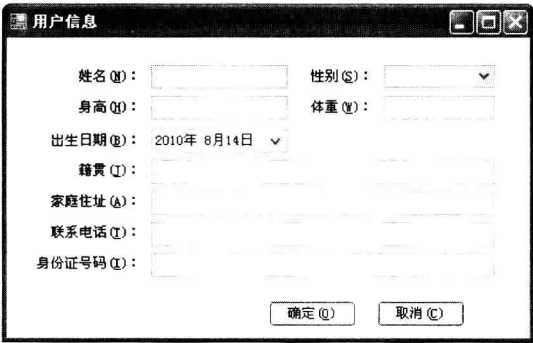


图 8-16 用户界面（右对齐）

在 VS.NET 中统一控件的对齐方式是比较简单的，首先将控件按照大致的位置布局好，然后再选中所有要参与对齐操作的控件，单击工具条上的对齐功能按钮即可。

图 8-17 就是 VS.NET 提供的对齐工具条。

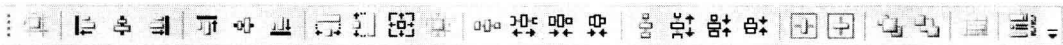







图 8-17 对齐工具条

该工具条上常用的按钮功能如表 8-2 所示。

表 8-2 工具条上常用的按钮功能

按 钮	功 能
左对齐	移动控件，当前控件不动，使得所有选择的控件的左边缘和当前控件对齐
居中对齐	移动控件，使得所有选择的控件的垂直中轴线和当前的控件重合
右对齐	移动控件，使得所有选择的控件的右边缘和当前的控件对齐
顶端对齐	移动控件，使得所有选择的控件的顶边缘和当前的控件对齐
中间对齐	移动控件，使得所有选择的控件的水平中轴线和当前的控件重合
底端对齐	移动控件，使得所有选择的控件的底边缘和当前的控件对齐
使宽度相同	设置所有选择的控件的宽度值等于当前控件的值
使高度相同	设置所有选择的控件的高度值等于当前控件的值
使大小相同	设置所有选择的控件的宽度和高度等于当前控件的值
使水平间距相等	适当地水平移动选择的控件，其中最左边的控件和最右边的控件不动，使得各个控件水平方向的间距相等
增加水平间距	适当地水平移动选择的控件，使得各个控件水平方向的间距增加一些，各个控件的间距增加量相同
减小水平间距	适当地水平移动选择的控件，使得各个控件水平方向的间距减少一些，各个控件的间距减少量相同
移除水平间距	适当地水平移动选择的控件，使得各个控件水平方向紧密地靠在一起，之间没有间隙
使垂直间距相等	适当地上下移动选择的控件，其中最上面和最下面的控件不动，使得各个控件垂直方向的间距相等

续表

按 钮	功 能
 增加垂直间距	适当地上下移动选择的控件，使得各个控件垂直方向的间距增加一些
 减小垂直间距	适当地上下移动选择的控件，使得各个控件垂直方向的间距减小一些
 移除垂直间距	适当地上下移动控件，使得各个控件垂直方向紧密地靠在一起，之间没有间隙
 置于顶层	使得选择的控件在 Z 顺序方向上变得靠前，不被其他控件遮盖
 置于底层	使得选择的控件在 Z 顺序方向上变得靠后，不会覆盖其他控件

有几个控件的位置设计如图 8-18 所示。

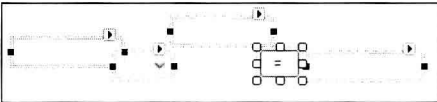


图 8-18 位置设计



此时需要选中这些控件，然后单击对齐工具条上的“ 中间对齐”和“ 移除水平间距”按钮，即可达到如图 8-19 所示的设计效果。



图 8-19 最终设计效果

8.5 Main 函数

C#中的 Windows 应用程序和命令行程序编译后的结果是 EXE 文件，而 EXE 文件需要定义启动程序的入口点。这个入口点在 C#中就是 Main 函数，这点和 C 语言很类似。

在“第一个 Windows 应用程序”的 C#工程中，一开始就包含了一个“Program.cs”的源代码文件，该文件内容如下：

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace 第一个 Windows 应用程序
{
    static class Program
    {
        /// <summary>
        /// 应用程序的主入口点
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles( );
            Application.SetCompatibleTextRenderingDefault(false);
```



```

        Application.Run(new Form1( ));
    }
}

```

在这个文件中定义了 Main 函数。一个 C# 的 Windows 应用程序或命令行程序中有而且只有一个 Main 函数。

在 Main 函数前面的代码 “[STAThread]” 的作用是通知编译器该函数是采用单线程模式的。

在此处 Main 函数没有定义参数和返回值，实际上 Main 函数可以定义一个 int 类型的返回值，还可以定义一个字符串数组类型的参数，如 “int Main(string[] args)”，这样可以获得命令行参数。一般情况下，具有图形用户界面的 Windows 应用程序不需要支持命令行参数，但命令行应用程序比较依赖命令行参数来获得调用者的意图。

除了在 Main 函数中获得命令行参数外，开发者还可以在任何地方使用类型 System.Environment 的 CommandLine 静态属性获得启动程序使用的命令行文本，也可以使用该类型的 GetCommandLineArgs 静态方法获得启动程序使用的命令行参数。

命令行文本是一个字符串，例如在 Windows 命令行界面中使用命令 “c:\my.exe *.jpg \a \s”，则 Environment 类型的 CommandLine 属性值为 “c:\my.exe *.jpg \a \s”，而它的 GetCommandLineArgs 方法返回一个有 4 个元素的字符串数组，数组值为 “c:\my.exe”、“*.jpg”、“\a”、“\s”。

Main 函数中的代码 “Application.EnableVisualStyles()” 的作用是让整个应用程序启动 XP 样式。

当 Windows 应用程序启动 XP 样式时，用户界面如图 8-20 所示。

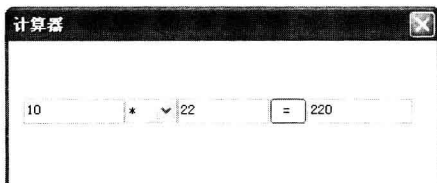


图 8-20 启动 XP 样式时的用户界面

当没有启动 XP 样式时，用户界面如图 8-21 所示。

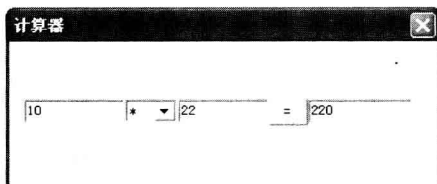


图 8-21 没有启动 XP 样式时的用户界面

Main 函数中的代码 “Application.Run(new Form1())” 的作用是创建一个窗体对象并运行该窗体。这个函数是同步执行的，只要窗体没有关闭，该函数就不会返回。当窗体关闭时，该函数返回，然后 Main 函数后面就没有代码了，于是程序退出。

8.6 解决方案资源管理器

VS.NET 集成开发环境右上方为解决方案资源管理器，其用户界面如图 8-22 所示。

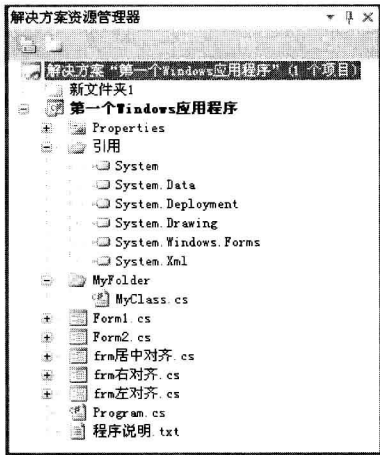


图 8-22 解决方案资源管理器


解决方案资源管理器包含一个工具条和一个解决方案资源树状列表。

8.7 解决方案资源树状列表

解决方案资源以树状列表的方式列出了 C# 工程中包含的所有成员项目。

8.7.1 解决方案

树状列表只有一个“解决方案”根节点，根节点下面列出了该解决方案下的所有成员项目。解决方案下面一般是各种类型的工程项目节点，还可以是其他文件。

在图 8-22 中，树状列表根节点是 “ 解决方案 ‘第一个Windows应用程序’ (1 个项目)”，它下面包含了一个 C# 工程节点。

用鼠标右击解决方案节点，会弹出如图 8-23 所示的快捷菜单。

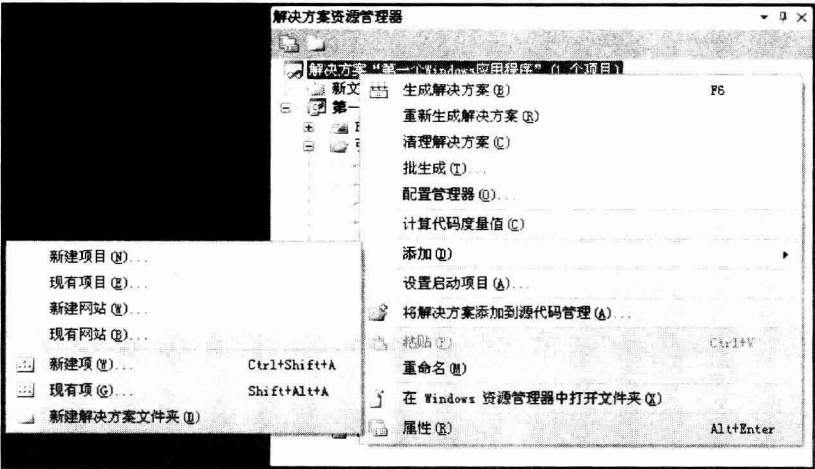


图 8-23 快捷菜单

该快捷菜单中常用的菜单项目如表 8-3 所示。

表 8-3 常用的菜单项目

菜 单 项 目	功 能
生成解决方案	对该解决方案下的所有程序工程进行编译，输出编译结果，所进行的是增量编译。在编译一个程序工程前会判断该工程及其内容是否发生改变，若改变了则进行编译，否则就用最后一个编译结果。这样能节省生成解决方案的执行时间 对于一个 C#工程，若在最后一次编译成功后，所有的程序源代码文件、资源文件没有被修改过，则生成解决方案时该 C#工程不会编译而使用最后一次的编译结果
重新生成解决方案	对解决方案下的所有程序工程进行编译，输出编译结果，但不会进行增量编译判断
添加	该菜单项目有子菜单项目
添加→新建项目	显示“添加新项目”对话框，若经用户的操作新增了一个项目，则将新增的项目添加到这个解决方案中。下一次打开解决方案时会加载这个刚刚新增的程序项目
添加→现有项目	显示一个“添加现有项目”的文件选择对话框，若用户选择了一个 VS.NET 支持的程序项目文件，比如 C#或 VB.NET 项目文件，则 VS.NET 加载该项目并添加到这个解决方案中。下一次打开解决方案时会加载这个程序项目
添加→新建网站	显示一个“添加新网站”对话框，若经用户的操作新增了一个网站类型的程序项目，则将这个新增的项目添加到这个解决方案中
添加→现有网站	显示一个“添加现有网站”对话框，若经用户的操作打开了一个网站类型的程序项目，则 VS.NET 将这个程序项目添加到解决方案中
重命名	修改解决方案的名称
属性	在 VS.NET 的主工作区域中显示解决方案属性窗口

8.7.2 程序工程

解决方案节点下列出了该解决方案中的程序工程节点。一个解决方案可以包含多个和多种程

序工程。比如一个解决方案可以包含若干个 C# 工程，若干个 VB.NET 工程或若干个 VC++.NET 工程，安装工程，等等。

在图 8-22 中，节点“ 第一个 Windows 应用程序”就是程序工程节点。

单击程序工程节点，会弹出如图 8-24 所示的快捷菜单。

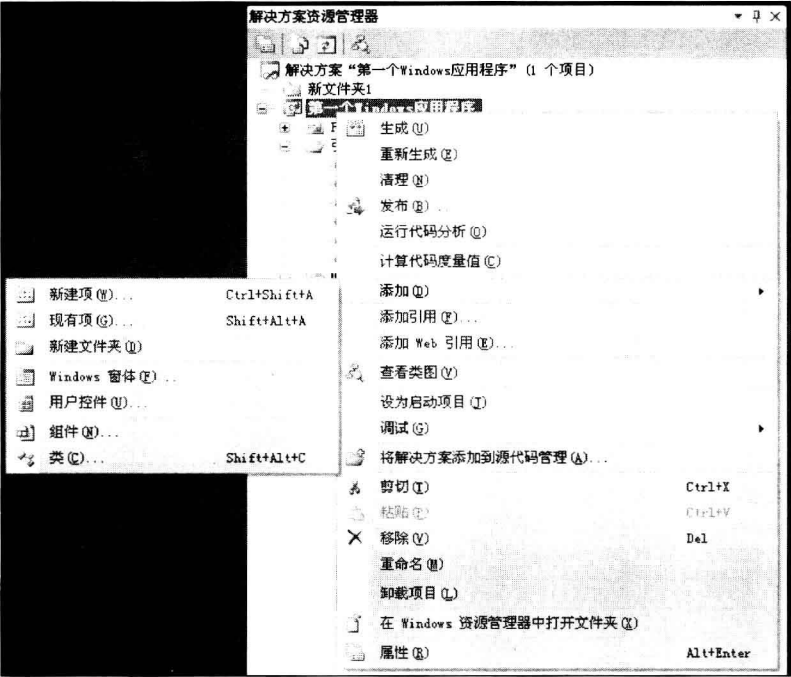


图 8-24 快捷菜单

该快捷菜单中常用的菜单项目如表 8-4 所示。

表 8-4 常用的菜单项目

菜单项目	功能
生成	对程序项目进行编译，输出编译结果。会进行增量编译判断，若最后一次生成后，所有的源代码或资源文件没有被修改过，则不会重新生成而直接使用上一次的编译结果
重新生成	对程序项目进行编译，输出编译结果。不会进行增量编译判断
添加	该菜单项目有若干子菜单项目
添加→新建项	显示“添加新项”对话框，可向这个程序项目中添加新的文件
添加→现有项	显示“添加现有项”对话框，可将多个已经存在的文件添加到程序项目中
添加→新建文件夹	在程序项目中新建一个文件夹
添加→Windows 窗体	显示“添加新项”对话框，并自动设置项目类型为“Windows 窗体”。用于快速地向程序项目中添加新的 Windows 窗体类型
添加→用户控件	显示“添加新项”对话框，并自动设置项目类型为“用户控件”。用于快速地向程序项目中添加新的用户控件类型

续表

菜单项目	功 能
添加引用	显示“添加引用”对话框，为程序项目添加一个程序集引用，使得软件能调用其他程序集中已经实现的软件功能
添加 Web 引用	显示“添加 Web 引用”对话框，为程序项目添加一个 Web 引用，使得软件能调用 WebService
设为启动项目	在 VS.NET 中开发人员单击 VS.NET 的菜单项目“调试→启动调试”，或者按下 F5 键以调试模式启动程序时，由于解决方案可以包含多个程序工程项目，而 VS.NET 不支持同时以调试模式运行多个程序项目，因此必须从多个程序工程项目中指定一个作为首先启动的项目。 使用本菜单项目就是指定当前程序工程项目为启动项目 启动项目必须为 WinForm 类型、命令行类型或网站类型，不能为类库类型
移除	在解决方案中删除该项目，但不会删除任何项目文件，只是让解决方案不再包含这个程序工程项目
重命名	修改程序工程项目的名称。项目工程名称和文件名是相同的，修改工程项目名称也会修改文件名。注意，修改工程名称不会修改工程的程序集名称和默认命名空间 在本例中可以看到，C#工程“第一个 Windows 应用程序”，它的工程文件名为“第一个 Windows 应用程序.csproj”；若修改项目工程名称为“第二个 Windows 应用程序”，则它的工程文件名应修改为“第二个 Windows 应用程序.csproj”。但这个工程编译输出结果还是“第一个 Windows 应用程序.exe”

8.7.3 引用

在树状列表的“引用”节点下列出了本 C#工程引用的外部程序集。外部程序集就是其他人已经开发好的独立的程序模块。开发人员在开发自己的程序中，可以调用这些现有的程序模块的功能，以实现软件的重用。

在本例中可以看到，C#工程“第一个 Windows 应用程序”引用了“System”、“System.Data”、“System.Deployment”、“System.Drawing”、“System.Windows.Forms”和“System.Xml”6 个外部程序集。

用鼠标右击“引用”节点会弹出如图 8-25 所示的快捷菜单。

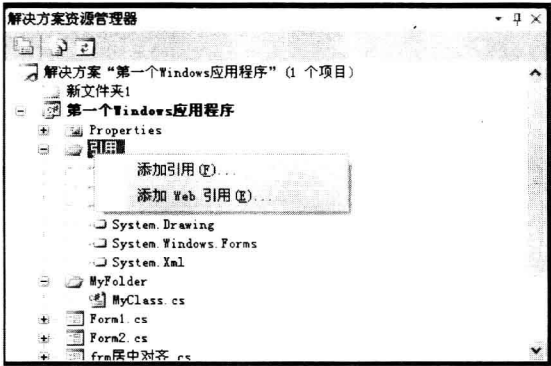


图 8-25 快捷菜单

该快捷菜单中，“添加引用”菜单功能等于程序工程的快捷菜单中的“添加引用”，“添加 Web 引用”菜单功能等于程序工程的快捷菜单中的“添加 Web 引用”。

8.7.4 程序文件

程序工程节点下大部分是程序文件节点，包括源代码文件、程序资源文件和其他文档文件。对于某些程序模块，比如窗体、用户控件、ASP.NET 页面等，需要多个程序文件的支持，此时会将这些相关联的文件显示为主文件的子节点。

一个窗体一般包含三个文件。比如对于窗体“frm 左对齐”，包含三个文件“frm 左对齐.cs”、“frm 左对齐.Designer.cs”和“frm 左对齐.resx”。其中第一个文件是窗体源代码主文件，“Designer.cs”是窗体设计器自动生成的代码文件，“resx”文件是窗体资源文件，用于保存窗体中的诸如图片等资源数据。

如图 8-26 所示，用鼠标右击文件节点，会显示一个快捷菜单。

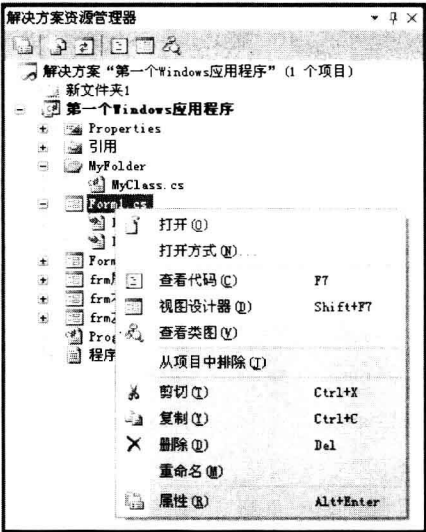


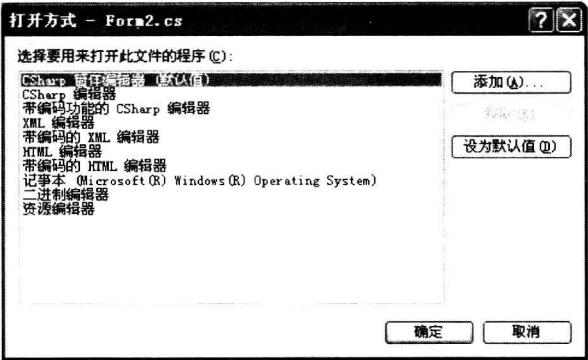
图 8-26 快捷菜单

该快捷菜单中常用的菜单项目如表 8-5 所示。

表 8-5 常用的菜单项目

菜单项目	功 能
打开	以与该文件类型对应的默认方式打开文件。在树状列表中用鼠标双击某个文件就相当于执行了“打开”操作 比如对于窗体代码文件，则在 VS.NET 主工作区域中打开窗体设计器 对于纯粹的源代码文件，则在主工作区域中显示该源文件的内容 对于 Word 文档，则会调用 MS Word 程序打开该文件

续表

菜单项目	功 能
打开方式	<div>显示如图 8-27 所示的“打开方式”对话框，选择某种打开方式即可打开文件</div> <div></div> <div>图 8-27 “打开方式”对话框</div>
查看代码	<div>直接查看文件的源代码，而不调用某种可视化的设计器</div> <div>比如对于窗体代码文件，默认为调用窗体设计器打开它，而单击“查看代码”则显示该窗体代码文件的 C#源代码</div>
视图设计器	<div>尽量调用与当前文件的文件类型配套的可视化设计器来查看和编辑文件内容</div> <div>比如对于本例中的 Form1.cs 调用窗体设计器来打开文件</div>
从项目中排除	<div>将当前文件从程序项目中去掉，使得程序项目不再包含该文件，但不会从磁盘中删除该文件。以后还可以用“添加现有项”的操作将文件重新包含在程序项目中</div>
剪切	剪切文件
复制	复制文件
删除	<div>删除文件。注意，该操作将从磁盘中永久地删除文件，而且无法使用 Windows 的回收站功能恢复文件</div>
重命名	修改文件名
属性	在属性列表中显示该文件的属性设置

对于不同的文件类型，该快捷菜单的内容可能会发生改变，即使对于同一种扩展名的文件其菜单内容也可能不同。

比如对于文件 Form1.cs，VS.NET 检测到该文件是一个窗体的主源代码文件，因此它的快捷菜单有“查看设计器”项目，而且单击“打开”菜单项目会调用 WinForm 窗体设计器来打开该源代码文件；而对于 Program.cs 文件，VS.NET 检测到该文件是一个纯粹的 C#源代码文件，此时它的快捷菜单就没有“查看设计器”，而且单击“打开”菜单项目会以文本方式显示该 C#源代码文件的内容。

8.7.5 文件夹

当程序文件非常多时，可以使用文件夹以树状的结构组织和管理程序文件，比如同属一个大

的功能模块的程序代码可以放在一个文件夹下。

对于 C# 工程，文件夹还会影响源代码的默认的命名空间。在项目中的子文件夹中新增程序源代码文件，则源代码中的命名空间就是“项目默认命名空间.子文件夹名”，而且子文件夹中的资源文件在编译后的程序集中的资源名称也是“项目默认命名空间.子文件夹名.资源文件名”。

对于本例中的 C# 工程“第一个 Windows 应用程序”，其默认命名空间是“第一个 Windows 应用程序”。在解决方案资源树状列表的 C# 工程项目节点上用鼠标右击，弹出快捷菜单，单击“添加→新建文件夹”菜单项目，此时会在工程节点下面新建一个“NewFolder1”的文件夹。

用鼠标右击“MyFolder”文件夹，会弹出如图 8-28 所示的快捷菜单。

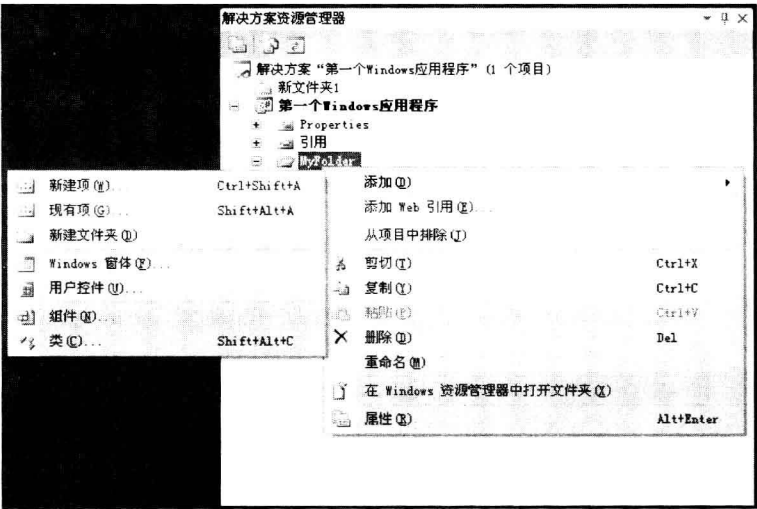


图 8-28 快捷菜单

该快捷菜单中常用的菜单项目如表 8-6 所示。

表 8-6 常用的菜单项目

菜单项目	功能
添 加	该项目有多个子菜单项目。这些子菜单项目用于在这个目录下添加新的文件或子文件夹
从项目中排除	从项目中排除这个文件夹以及所有的子文件夹和文件，但不从磁盘中删除该文件夹及其内容
重 命 名	修改该文件夹的名称

单击“重命名”菜单项目将该文件夹名称修改为“MyFolder”，这样就完成了新增一个指定名称的文件夹的操作。

在 MyFolder 文件夹下面新增一个名为“MyClass”的 C#类源代码文件。打开该文件可以看到其代码文本如下：

```
using System;
using System.Collections.Generic;
using System.Text;
```



```
namespace 第一个 Windows 应用程序.MyFolder
{
    class MyClass
    {
    }
}
```

默认情况下，这个类型的命名空间就是“第一个 Windows 应用程序.MyFolder”。


注意，在修改该文件夹的名称后，该文件夹下已有的源代码的命名空间不会自动更新，但此后新增的源代码中的命名空间包含了新的文件夹名称。

开发人员可以在文件夹下再建立子文件夹，形成多级目录结构。此时在多层文件夹下面新增的 C#代码文件其默认的命名空间就是“项目默认命名空间.文件夹名.子文件夹名”，以此类推。这种文件夹层级关系理论上没有限制，但在实践中至多用 4 层。

8.8 解决方案资源管理工具条

解决方案资源树状列表上面是工具条，工具条上的各个按钮功能如下。

8.8.1 属性按钮

开发人员单击“属性”按钮可以打开树状列表中当前项目的属性窗口。一般都会切换焦点到解决方案资源管理器下面的属性编辑区域。但当前项目是某个工程根节点，比如在图 8-28 中的“ 第一个 Windows 应用程序”，则会在 VS.NET 的主工作区域中显示如图 8-29 所示的项目属性窗口。

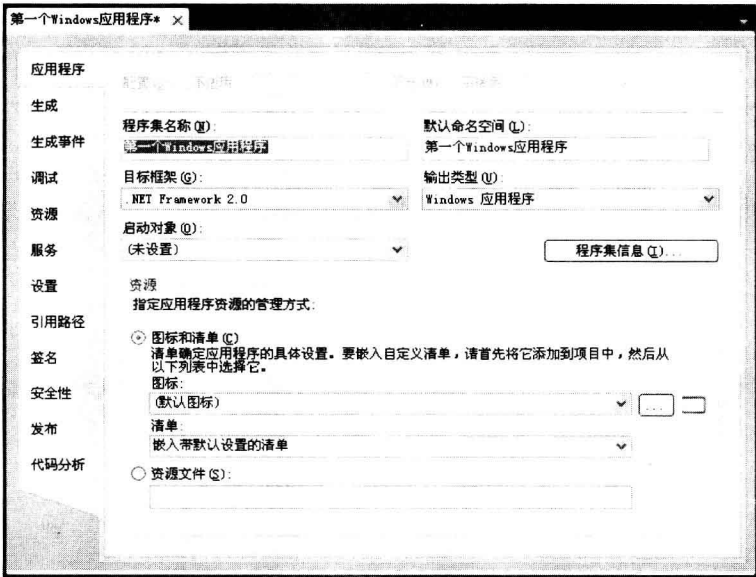



图 8-29 项目属性窗口

在项目属性窗口中，“程序集名称”文本框中输入的是编译后的程序集文件名，可以为“程序集名称.EXE”或“程序集名称.DLL”。“默认命名空间”文本框中输入的是该 C# 项目中的根命名空间。在这个项目中新增的类型的命名空间就是“默认命名空间.文件目录名.类型名称”。“目标框架”可以选择编译后的程序集运行在哪个版本的.NET 框架之上，可以选择“.NET2.0”、“.NET3.0”、“.NET3.5”、“.NET4.0”等。“输出类型”下拉列表列出了程序集类型，可选择的类型有“Windows 应用程序”、“控制台应用程序”、“类库”。第一、第二种类型，编译为 EXE 文件，对于类库类型编译为 DLL 文件。

若树状列表中当前节点是解决方案节点，比如“ 解决方案‘第一个Windows应用程序’”，则程序会弹出如图 8-30 所示的解决方案属性页对话框。

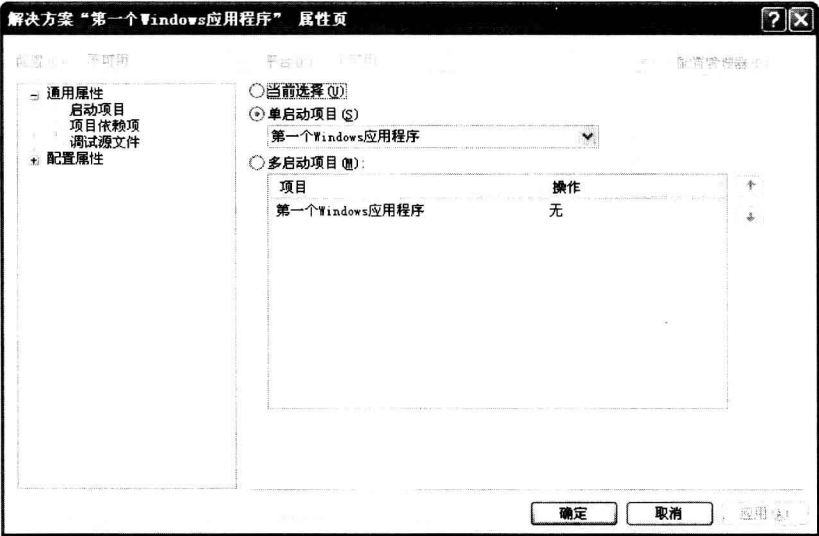


图 8-30 解决方案属性页对话框

若树状列表中当前节点是某个程序文件，比如“Form1.cs”，则单击“属性”按钮，用户焦点会切换到控件属性编辑区域，此时的属性列表如图 8-31 所示。



图 8-31 属性列表

窗体文件的主要属性如表 8-7 所示。

表 8-7 窗体文件的主要属性

属 性	说 明
复制到输出目录	<p>该属性可以选择“不复制”，“始终复制”和“如果较新则复制”三种选项。各个选项意义如下：</p> <p>不复制：在 VS.NET 编译程序后，不会将这个文件复制到编译结果输出目录下。比如这个 C#工程编译后在工程所在文件的“bin\Debug”子目录下生成“第一个 Windows 应用程序.exe”可执行文件，则 Form1.cs 文件不会复制到这个可执行文件的目录下</p> <p>始终复制：在 VS.NET 成功编译程序后将该文件复制到编译结果输出目录下。比如本 C#工程编译后在“bin\Debug”子目录下生成 EXE 文件，则还会将 Form1.cs 文件复制到这个可执行文件的目录下</p> <p>如果较新则复制：在 VS.NET 成功编译程序后在输出目录下生成编译结果。然后判断输出目录下的文件是否是最新的，若不是最新的则复制文件。比如在本 C#工程编译后在“bin\Debug”子目录下生成 EXE 文件，然后判断输出目录下是否有 Form1.cs 文件以及文件的建立时间。若工程目录下的 Form1.cs 文件比输出目录下的 Form1.cs 文件要新，则将工程目录下的 Form1.cs 文件覆盖掉输出目录下的文件</p> <p>一般将 C#工程中的基础数据文件等应用程序需要使用的文件设置为“始终复制”，其他文件都设置为“不复制”</p>
生成操作	<p>该属性可以选择“无”，“编译”，“内容”和“嵌入的资源”四种选项。各个选项的意义如下：</p> <p>无：该文件仅包含在工程文件中，在程序编译或打包时不做任何处理</p> <p>编译：该文件是程序源代码文件，参与程序的编译。比如扩展名为 cs 的 C#源代码文件</p> <p>内容：该文件包含在工程文件中，不参与编译，但在对程序进行打包生成安装文件时，会将该文件包含在安装包中。当程序安装后，该文件也会复制到程序安装目录下</p> <p>嵌入的资源：该文件作为程序资源文件，参与程序的编译。该文件的内容将作为程序资源嵌入在程序编译输出文件中。比如可以将程序要使用的光标、图标等文件设置为嵌入的资源</p>
完整路径	该文件的绝对路径名

注意，窗体文件属性和窗体属性是不同的，窗体文件属性是针对定义窗体的文件的一些属性设置，内容很简单，而窗体属性是针对窗体对象的属性，内容很丰富。

在本例中，打开 Form1 的窗体设计界面，单击窗体则属性列表会显示该窗体对象的属性，如图 8-32 所示。若在解决方案资源树状列表中选中文件 Form1.cs，则属性列表会显示该窗体源代码文件的属性，如图 8-33 所示。经过对比可以看出存在明显的差别。



图 8-32 显示窗体对象的属性



图 8-33 显示窗体源代码文件的属性

8.8.2 添加新解决方案文件夹按钮

在解决方案资源树状列表中选中解决方案节点，则工具条上只有两个按钮，第一个为“属性”按钮，第二个为“添加新解决方案文件夹”按钮，如图 8-34 所示。

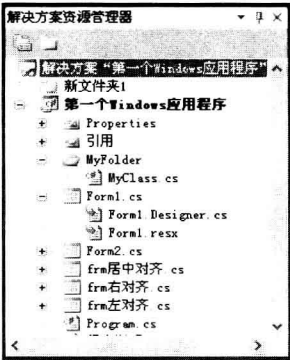


图 8-34 解决方案资源管理器

单击“添加新解决方案文件夹”按钮，系统会在解决方案下面新增一个文件夹，可以缓慢地双击新增的文件夹或单击文件夹，在弹出的快捷菜单中单击“重命名”菜单项目来修改文件夹的名称。

在开发实践中，基本上没有直接在解决方案下新增文件夹或文件的，因此不做详细描述。

8.8.3 显示所有文件按钮

在解决方案资源树状列表中选中程序项目节点或其任意子节点，工具条上的第二个按钮就是“显示所有文件”按钮。如图 8-35 所示为没有按下“显示所有文件”按钮时的用户界面，图 8-36 为按下“显示所有文件”按钮后的用户界面。

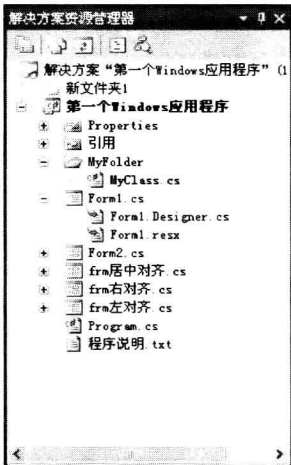


图 8-35 用户界面 1



图 8-36 用户界面 2

按下“显示所有文件”按钮，则该项目的树状结构中列出了项目文件所在文件夹中的所有文件和文件夹结构，此时那些并不包含在项目中的文件和文件夹也显示出来了，但图标比较淡。

用鼠标右击那些不包含在项目中的文件或文件夹，弹出如图 8-37 所示的快捷菜单。

单击该快捷菜单中的“包括在项目中”菜单项目，若选中的是文件，则将该文件包含在项目中；若选中的是文件夹，则将这个文件夹及所有的子孙文件夹和文件包含在项目中。

具体做法是在 Windows 资源管理器中将所需的文件复制到工程项目目录下，然后在 VS.NET

中单击“显示所有文件”按钮，在解决方案资源树状列表中选中文件，然后使用“包括在项目”中的快捷菜单项目将文件包含在项目中。这样比前面介绍的“添加→现有项”操作要快得多。

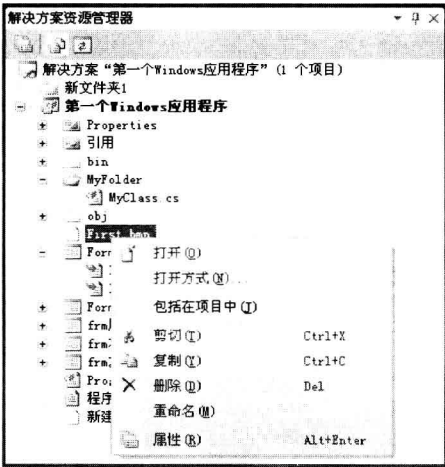


图 8-37 快捷菜单

8.8.4 刷新按钮

在解决方案资源树状列表中选中程序项目节点或其任意子节点，工具条上的第三个按钮就是“刷新”按钮。

当其他程序或用户在 Windows 资源管理器中修改了解决方案资源树状列表中列出的文件后，可以单击这个“刷新”按钮来刷新解决方案资源树状列表中的状态。

在本例中，项目包含了一个“程序说明.txt”的文件，则解决方案资源管理树状列表界面如图 8-38 所示。当用户在 Windows 资源管理器中将这个文件删除时，VS.NET 并没有感觉到这个文件被删除了，只有单击“刷新”按钮，VS.NET 才感觉到文件被删除了，此时已经遭到删除的项目文件节点图标带有黄底色的惊叹号标记，如图 8-39 所示。

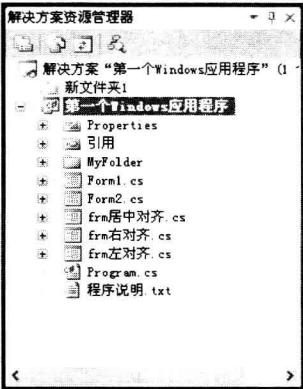


图 8-38 树状列表界面 1

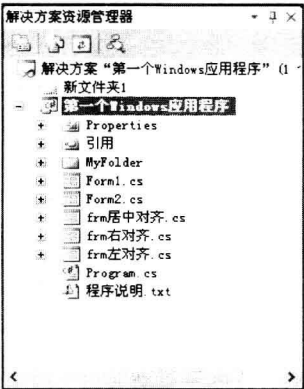


图 8-39 树状列表界面 2

8.9 控件属性编辑区域

在 VS.NET 的窗体设计界面的右下方为控件属性编辑区域。开发人员可以在这个区域中查看和编辑窗体设计器中选中的控件的一些属性。

如图 8-40 所示，属性编辑区域分为控件名称下拉列表、属性列表工具条、编辑区域（其中间大部分区域是属性项目列表）和说明区域四个部分。

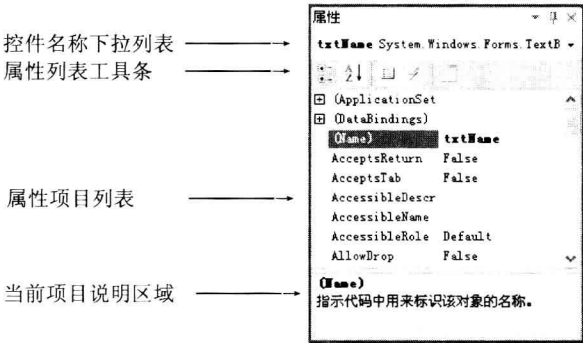


图 8-40 属性编辑区域

8.9.1 控件名称下拉列表

控件名称下拉列表列出了窗体中所有控件的名称和类型，其界面如图 8-41 所示。

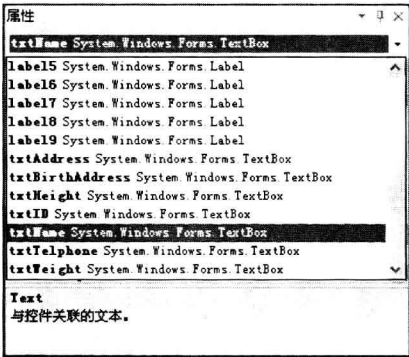


图 8-41 控件名称下拉列表

该列表只是简单地列出所有的控件，并没有体现出控件的层次关系。开发人员可以通过选中该列表中的项目来选中窗体中的某个控件。当窗体比较复杂，存在控件的叠加使某些控件被其他控件遮盖而看不到因此选不中时，开发人员可以使用这个控件名称下拉列表来选中某个控件。此时属性列表编辑区域可显示当前控件的所有可设计的属性。

从该列表的内容可以看到设置控件名称的重要性，若没有重新设置控件的名称，则列表中

列出类似 `textBox1`、`textBox2`、`textBox3` 之类的无意义的名称，难于快速查找和选择指定功能和意义的控件。

8.9.2 属性列表工具条

属性列表工具条列出了一些设置属性列表显示样式的工具按钮，对于大多数控件有 4 个可用的按钮。

(1) 按分类排序按钮

如图 8-42 所示，若用户按下该按钮，则属性列表中按照控件属性的分类进行排序。

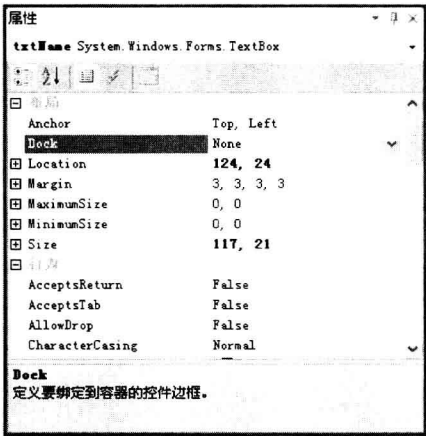


图 8-42 属性列表 1

在开发控件时可以声明其属性属于某种类型。这样当属性列表处于分类排序的显示方式时，各个属性可按类集中在一起，然后在类型中按名称进行排序。属性的类型只是一个面向开发人员的附加的说明，对程序的功能没有任何影响，因此编写代码时无须考虑它。

常见的属性类别如表 8-8 所示。

表 8-8 常见的属性类别

属 性 类 别	说 明
操作	操作相关的属性
外观	与控件外观相关的属性，比如 <code>ForeColor</code> 、 <code>BackColor</code> 等
行为	与控件运行时的行为相关的属性，比如 <code>Multiline</code> 等
数据	与数据以及数据源管理相关的属性，比如 <code>Tag</code> 、 <code>DataSource</code> 等
设计	仅在设计时可用的属性，比如 <code>Name</code> 等
布局	与控件布局相关的属性，比如 <code>Left</code> 、 <code>Top</code> 、 <code>Width</code> 、 <code>Height</code> 等
杂项	所有没有声明属于某种类别的属性都划归到杂项类别下

开发人员在开发自定义用户控件时可以自己定义新的属性类别。

(2) 字母顺序按钮

属性列表工具条的第二个按钮是字母顺序按钮。如图 8-43 所示，按下该按钮则控件的属性不再按照分类区别开来，而是统一按照属性名字母顺序从小到大地进行排列显示。

使用这种排序方式，开发人员可以凭着对控件属性名称的一些模糊记忆来快速查找所需的属性项目。这是对控件比较熟悉的开发人员所使用的排序方式。

(3) 属性按钮

属性列表工具条的第三个按钮是属性按钮。若按下该按钮，则属性列表中列出的是控件的可设计的属性。在上面的两个截图里，属性按钮都处于按下状态。

(4) 事件按钮

属性列表工具条的第四个按钮是事件按钮。若按下该按钮，则属性列表中列出的是控件的事件，如图 8-44 所示。

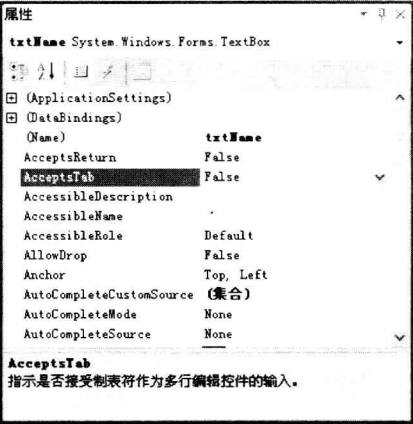


图 8-43 属性列表 2



图 8-44 属性列表 3

当同时按下按分类排序按钮和事件按钮时，属性列表如图 8-45 所示。

在事件列表中双击某个事件项目时，程序会切换到窗体的源代码编辑界面中，自动生成一个名为“控件名称_事件名称”的函数，用户可以在这个新增的函数中输入响应事件的程序代码。

在图 8-45 中，若开发人员选中了名为“txtName”的文本框，并在事件属性列表中双击“TextChanged”项目，则程序会自动生成一个名为“txtName_TextChanged”的函数。

若窗体代码中已经有符合事件委托类型的函数，则开发人员可以单击事件右边的下拉按钮显示一个下拉列表，然后将某个已经存在的函数绑定到控件事件上。

如图 8-46 所示，若开发人员已经为其他的 TextBox 控件编写了 TextChanged 事件处理函数，则单击下拉按钮可显示一个列表，该列表列出了窗体程序代码中所有可绑定到 TextBox 控件的 TextChanged 事件的函数。开发人员可以选择一个函数，这样一个事件处理函数就能响应多个控件的事件。这样做在一些情况下是有用的，能减少代码量并在一个函数中进行事件的集中响应

处理。

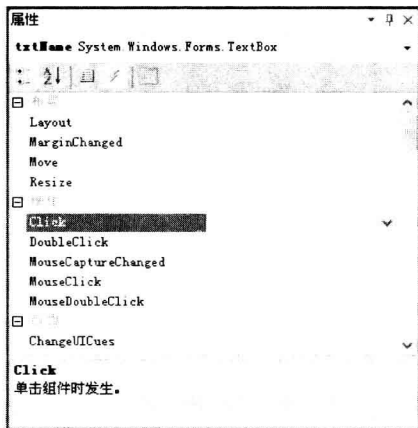


图 8-45 属性列表 4

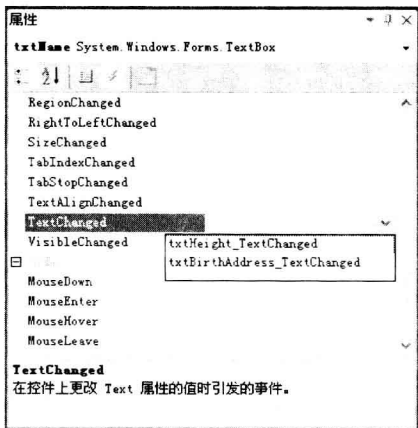


图 8-46 属性列表 5

8.9.3 属性项目列表

控件属性编辑区域中间大部分区域是属性项目列表，该列表列出了当前选中的控件的所有可设计的属性。其内容和现实方式受属性列表工具条上按钮的设置的影响。该列表用户界面如图 8-47 所示。



图 8-47 属性项目列表

该列表左边为属性名称，右边为属性值。若属性的数据类型为复合类型，则属性项目前有一个收缩展开图标，开发人员可以单击这个图标来展开和收缩现实属性值的子属性。这种现实方式是可以套嵌的。比如 TextBox 控件的 Location 属性值是 System.Drawing.Point 类型，Point 类型有 X 和 Y 属性，因此开发人员可以单击 Location 项目前的图标来展开或收缩该属性值的 X 和 Y 的子属性。

属性列表中一般用纯文本显示属性值，有时也会在属性值前面用一个小矩形区域辅助显示属性值。比如 TextBox 控件的 ForeColor 属性表示文本颜色，在属性列表的颜色属性值前会显示一

个小方框，方框的颜色就是当前设置的颜色值。

某些属性值是用粗体字显示的，这是为了突出显示当前设置的属性值不等于该属性的默认值。比如 `TextBox` 控件的 `ForeColor` 属性默认值为 `WindowText`，当设置属性值为 `WindowText` 时以正常字体显示属性值文本。如图 8-48 所示，当设置 `ForeColor` 属性值为其他颜色，比如“Blue”时，属性值不等于默认值，此时会以粗体字突出显示属性值。

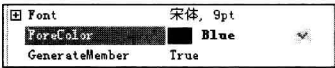


图 8-48 属性值

开发人员在开发控件时可以指定属性的默认值。以下代码为 `Multiline` 属性指定的默认值为 `false`。

```
private bool bolMultiline = false;
[System.ComponentModel.DefaultValue( false )]
public bool Multiline
{
    get { return bolMultiline; }
    set { bolMultiitline = value; }
}
```

(1) 扩展属性编辑器

开发人员单击属性项目列表右边的属性值显示区域，会进入属性值编辑模式，此时开发人员可以输入单行文本来编辑属性值。对于大多数属性，开发人员可以直接编辑属性值，但可能有一部分属性不能直接编辑，而需要使用扩展属性编辑器。

开发人员在属性项目列表中选中某个属性时，在该项目的右边可能会出现一个小按钮，也可能什么都没有。比如对于 `TextBox` 控件，不同的属性在属性列表中的显示样式有以下三种。

- 普通样式

如图 8-49 所示，属性项目右边没有任何按钮，此时开发人员只能在属性值输入框中直接修改属性值。

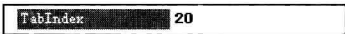


图 8-49 属性值输入框

- 下拉编辑样式

如图 8-50 所示，属性项目右边出现一个下拉按钮，此时开发人员可以通过单击这个下拉按钮来弹出另外的用户界面，以便编辑属性值。



图 8-50 属性项目

开发人员单击下拉列表弹出的用户界面不一定是列表，而是多种多样丰富多彩的。图 8-51

显示了编辑 TextBox 控件的某些属性时弹出的用户界面。

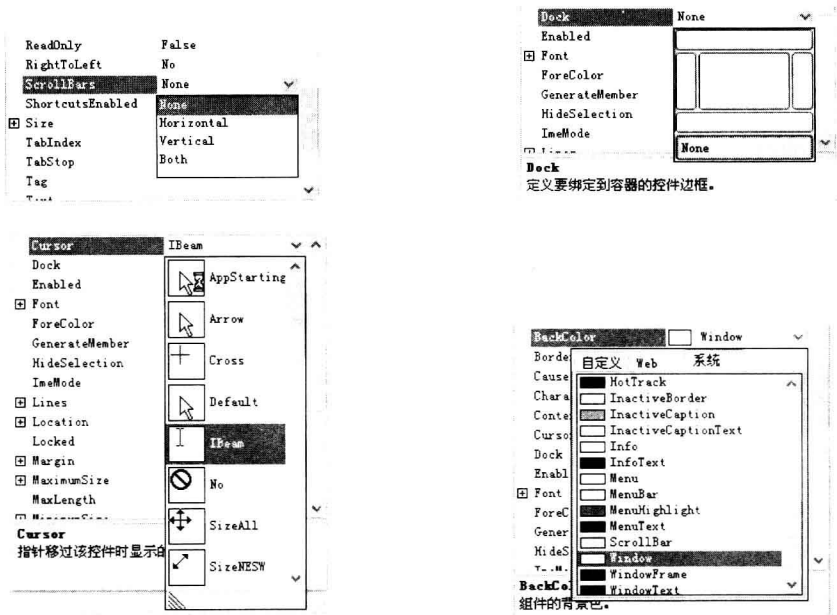


图 8-51 用户界面

开发人员开发自定义组件时可以为组件的属性定义自己的下拉样式的属性值编辑界面。

- 弹出对话框样式

如图 8-52 所示，有些属性项目右边显示一个带三个点的按钮。当开发人员单击这个按钮时，程序会弹出一个对话框，以便编辑属性值。



图 8-52 属性项目

例如对于 TextBox 控件的 Font 属性，单击字体属性项目后面的小按钮，就能弹出如图 8-53 所示的对话框，以便编辑属性值。

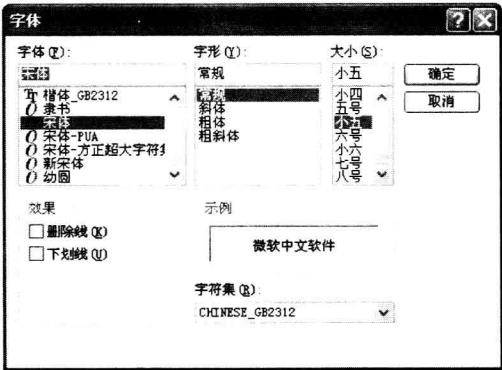


图 8-53 “字体”对话框

开发人员可以在开发自定义组件时为组建属性定义自己的对话框来编辑属性值。

8.10 设计用户界面

上面讲的都是如何使用 VS.NET 来设计用户界面，现在利用上面学习的知识来设计第一个 Windows 应用程序的用户界面。

前面提到，本程序的功能就是做一个能进行加、减、乘、除的计算器。

当新建一个 Windows 窗体应用程序项目时，项目中已经存在一个名为“Form1.cs”的窗体。为了规范起见，笔者将该窗体改名为“frmCalculate”。操作很简单，在解决方案资源管理器中，选中节点“Form1.cs”，然后用鼠标右击弹出快捷菜单，单击“重命名”菜单项目，然后修改窗体文件名“Form1.cs”为“frmCalculate.cs”，此时 VS.NET 会弹出如图 8-54 所示的对话框。

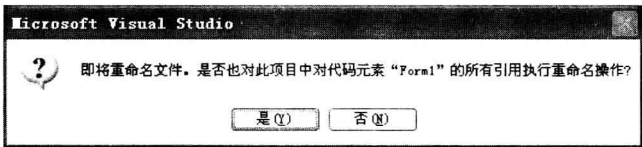


图 8-54 提示对话框

VS.NET 在修改源代码文件名时会自动检测该源代码文件中的类型信息，若类型名称等于文件名，则 VS.NET 提供一次机会，能将类型名称也改成新的名称。

例如在本例中，源代码文件“Form1.cs”中定义了一个类型“Form1”，当将文件“Form1.cs”改名成“frmCalculate.cs”时，系统会提供一次机会修改类型“Form1”为“frmCalculate”。这种替换是遍及整个解决方案的，如果其他文件中的源代码引用了“Form1”类型，也一并修改成“frmCalculate”。这样改名后，整个应用程序就能正确地编译了。

在该对话框中，一般情况下应该选择“是”，以便进行类型改名。


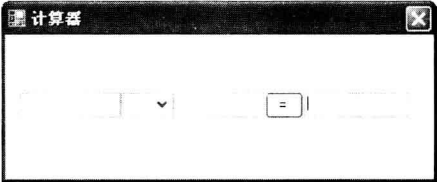
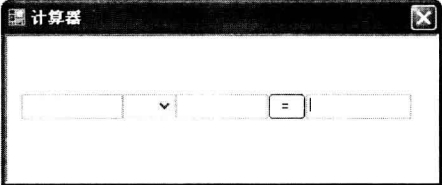
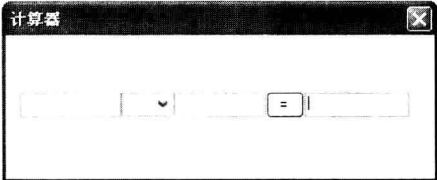
在解决方案资源管理器中双击“frmCalculate”节点，打开窗体设计界面。在属性编辑器中设置窗体对象的属性，设置的属性值如表 8-9 所示。

表 8-9 属性值

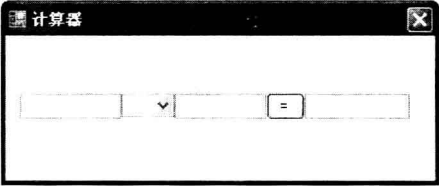
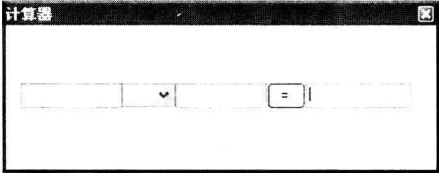
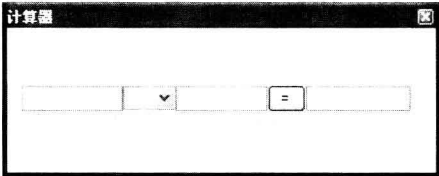
属 性 名	属 性 值	说 明
FormBorderStyle	FixedDialog	设置窗体的边框样式。该属性值类型是 System.Windows.Forms. FormBorderStyle 枚举类型，其可选值可参照表 8-10，该属性默认值为“Sizable”
MaximizeBox	False	不显示标题栏上的最大化窗体按钮
MinimizeBox	False	不显示标题栏上的最小化窗体按钮
StartPosition	CenterScreen	窗体默认的显示位置
Text	计算器	窗体标题栏文本

表 8-10 是窗体对象的 FormBorderStyle 在不同的选项值下的外观。

表 8-10 FormBorderStyle 值及说明

FormBorderStyle 值	说 明
None	<p>无边框。没有标题栏和关闭按钮，用户不能拖曳边框来移动窗体或改变窗体的大小，而只能使用“Alt+F4”组合键来关闭窗体，如图 8-55 所示</p> <div></div> <p>图 8-55</p>
FixedSingle	<p>固定的单行边框。用户可以拖曳标题栏来移动窗体，但不能拖曳边框来改变窗体的大小，如图 8-56 所示</p> <div></div> <p>图 8-56</p>
Fixed3D	<p>固定的三维边框，窗体容器有三维下陷样式的粗边框。用户可以拖曳标题栏来移动窗体，但不能拖曳边框来改变窗体的大小，如图 8-57 所示</p> <div></div> <p>图 8-57</p>
FixedDialog	<p>固定的对话框样式的粗边框。标题栏没有图标，如图 8-58 所示</p> <div></div> <p>图 8-58</p>

续表

FormBorderStyle 值	说 明
Sizable	<p>可调整大小的边框。用户可以拖曳窗体的边框来改变窗体的大小，如图 8-59 所示</p> <div></div> <p>图 8-59</p>
FixedToolWindow	<p>不可调整大小的工具窗口边框。标题栏比较矮，并且无图标，用户不能改变窗体的大小，如图 8-60 所示</p> <div></div> <p>图 8-60</p>
SizableToolWindow	<p>可调整大小的工具窗口边框。标题栏比较矮，并且无图标，用户可以改变窗体的大小，如图 8-61 所示</p> <div></div> <p>图 8-61</p>

设置好窗体属性后，从工具箱上拖曳几个控件到窗体上，然后调整控件的位置和大小，此时窗体设计如图 8-62 所示。

从左到右，依次设置控件的名称为“txtNumber”、“cboOperator”、“txtNumber2”、“btnCalculate”、“txtResult”。

选中控件 cboOperator，在属性编辑器中设置 DropDownStyle 属性值为 DropDownList。

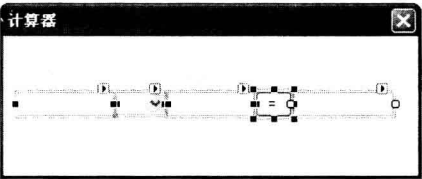

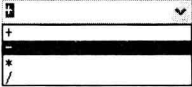



图 8-62 窗体设计

组合框控件的类型为“System.Windows.Forms.ComboBox”，它的 DropDownStyle 属性值类型为 ComboBoxStyle 枚举类型，其可选值如表 8-11 所示。

表 8-11 属性值

属 性 值	说 明
Simple	<p>此时控件显示出一个文本框，文本框下面有一个可选值的列表，用户可以直接输入任意文本数据，也可以在列表选择一个数据。其用户界面如图 8-63 所示</p>  <p>图 8-63</p>
DropDown	<p>此时控件是可以直接输入任意数据的，也可以单击向下箭头显示一个弹出式的下拉列表，从中选中一个数据。其用户界面如图 8-64 所示</p>  <p>图 8-64</p>
DropDownList	<p>此时控件不让用户直接输入数据，而需要单击控件显示一个下拉列表，用户只能在这个列表中选择某个数据项目。其用户界面如图 8-65 所示</p>  <p>图 8-65</p>

设置完 DropDownStyle 属性后，选中 Items 属性，单击属性值后面的小按钮，弹出如图 8-66 所示的对话框，输入需要支持的运算符，然后单击“确定”按钮关闭该对话框。还应设置 txtResult 控件的 ReadOnly 属性值为 True。

到此窗体设计完毕，此时单击 VS.NET 主菜单项目“调试→启动调试”，或者按下快捷键 F5 运行应用程序，可以看到其用户界面如图 8-67 所示。

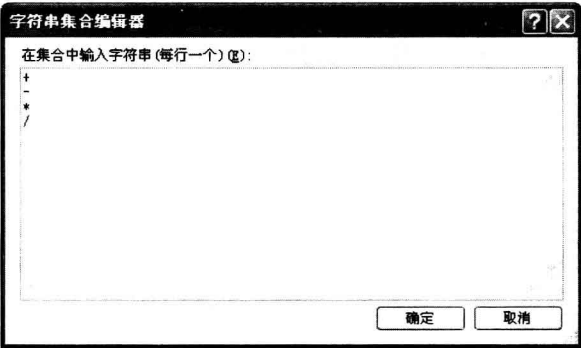


图 8-66 “字符串集合编辑器”对话框

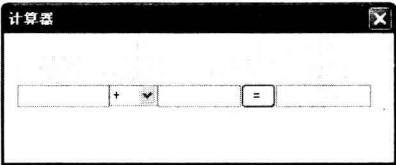


图 8-67 用户界面

8.11 用户界面事件处理原理

上面使用强大的 WinForm 窗体设计器设计出应用程序的用户界面，但这个用户界面是固定的，对任何操作都没有反应。此时我们需要给这个用户界面编写后台代码来实现一些程序功能，让应用程序能实现灵活的功能。

WinForm 应用程序通常是用户操作驱动的，也就是说，用户需要用鼠标和键盘等操作来调用程序功能，若用户没有任何操作，程序就静静地呆在那里等候用户的操作。

在 .NET WinForm 程序中，任何用户的鼠标和键盘操作都会映射为控件的事件。下面介绍 .NET 开发中用户界面事件处理模型。

计算机用户经常操作鼠标和键盘来试图干预和控制应用程序的运行，而应用程序需要感知鼠标和键盘的状态来响应用户的操作意图。此时开发人员需要进行鼠标和键盘事件处理。图 8-68 是 .NET 应用程序处理鼠标和键盘事件的原理图

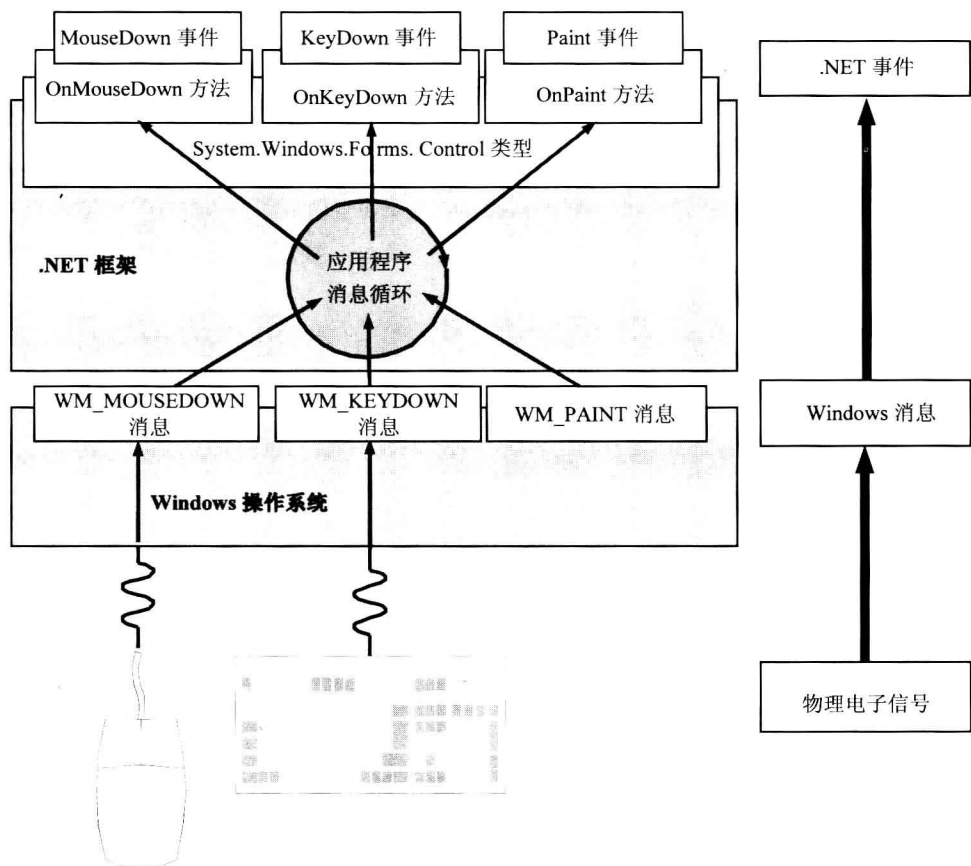


图 8-68 原理图

依据这个原理图，用户使用鼠标来操作程序的整个过程如下：

(1) 用户用手单击鼠标按键。

(2) 鼠标是一个固体电子设备，当用户按下鼠标按键时，鼠标的内部电路会发出一个电子脉冲，通过鼠标电缆发送到计算机主机中。

(3) 计算机主机的 CPU 接收到这个电子脉冲后，调用操作系统的处理鼠标的相关程序模块。

(4) 操作系统接收鼠标信息，并转换为一个名为“WM_MOUSEBUTTONDOWN”的 Windows 消息，然后发往应用程序。

(5) Windows 应用程序都具有一个 Windows 消息循环处理的功能，用于不停地处理所有接收的 Windows 消息。而在 .NET 应用程序中，.NET 框架自动处理 Windows 消息循环，并根据不同的消息调用控件（System.Windows.Forms.Control 类型）的不同方法。在这里 .NET 框架接收“WM_MOUSEBUTTONDOWN”消息，然后调用控件的 OnMouseDown 方法。

(6) 控件默认的 OnMouseDown 方法触发控件的 MouseDown 事件。

(7) 开发人员开发自定义的控件是为了重载控件的 OnMouseDown 方法或响应控件的 MouseDown 事件。当重载控件的 OnMouseDown 方法时，根据面向对象的编程原理，.NET 框架调用控件的 OnMouseDown 方法时实际上调用了开发人员编写的 OnMouseDown 方法，从而执行开发人员编写的代码。

同样地，键盘也是一个固体电子设备，用户按下某个按键时键盘的内部电路会向计算机主机发送一个电子脉冲，形成键盘硬件消息，而操作系统会将键盘硬件消息加工成名为“WM_KEYDOWN”的 Windows 消息，然后发送到应用程序消息处理循环队列中。.NET 框架处理“WM_KEYDOWN”消息时会转而调用控件的 OnKeyDown 方法，而开发人员可以重载控件的 OnKeyDown 方法来处理键盘事件。

在这个过程中，通过计算机主机、操作系统和 .NET 框架的幕后协同工作，实现了物理电子信号到 Windows 消息到 .NET 事件的三步转换过程，最后使开发人员仅仅通过重载控件的方法或响应控件的事件，即可处理一些常见的硬件操作事件，这样不仅降低了 Windows 程序开发难度，而且还提高了开发效率。

这里的窗体重绘消息“WM_PAINT”不是源自于硬件操作，而是操作系统自己产生的。当用户操作发生了诸如窗体被遮盖后重新显示等情况时，操作系统会自行生成“WM_PAINT”消息发往应用程序；而 .NET 框架会处理“WM_PAINT”消息，然后调用控件的 OnPaint 方法或触发 Paint 事件，开发人员只需重载控件的 OnPaint 方法和响应 Paint 事件，即可完成控件用户界面的重新绘制。

8.11.1 鼠标事件

大部分情况下用户要求能用鼠标操作来进行处理，此时软件必须响应用户的鼠标操作来进行某些处理。在 C# 的窗体软件开发中，开发人员需要响应控件的鼠标事件，此时需要绑定控件的

MouseDown 事件或重写 OnMouseDown 方法来处理鼠标按键按下事件，绑定控件的 MouseMove 事件或重写 OnMouseMove 方法来处理鼠标移动事件，绑定控件的 MouseUp 事件或重写 OnMouseUp 方法来处理鼠标按键松开事件，绑定控件的 MouseWheel 事件或重写 OnMouseWheel 方法来处理鼠标滚轮事件。

8.11.2 键盘事件

用户经常要求能用键盘来进行操作，比如对于文本编辑器，用户需要使用键盘来输入大量的文本字符，移动光标等。此时软件必须响应用户的键盘操作来进行某些处理。在 C# 窗体软件开发中，开发人员需要响应控件的键盘事件，此时需要绑定控件的 KeyDown 事件或重写 OnKeyDown 方法来处理键盘按键按下事件，绑定控件的 KeyUp 事件或重写 OnKeyUp 方法来处理键盘按键松开事件，绑定控件的 KeyPress 事件或 OnKeyPress 方法来处理键盘字符事件。

KeyDown 事件和 KeyPress 事件是有区别的，用户按下键盘上的任何按键都会触发 KeyDown 事件，包括字符键、功能键、光标键等，该事件的参数是按键的编码。用户只有按下键盘的字符键才会触发 KeyPress 事件，此时该事件的参数就是按键的字符数据（C# 中的 char 类型）。此外用户使用各种输入法输入的非英文字符也会触发 KeyDown 事件，比如用户使用中文输入法输入一个汉字，则会触发一个 KeyPress 事件，事件的参数就是该汉字的编码。

8.12 编写事件处理代码

学习了用户界面事件处理原理后，就可以为窗体添加后台代码来实现一些功能。

已经设计好的窗体如图 8-69 所示。我们只需处理“=”按钮的点击事件就完功能了。这个功能很简单，就是将文本框 txtNumber1 和 txtNumber2 中用户输入的文本转换为数值，然后根据 cboOperator 控件中用户选择的运算符进行数学运算，最后将运算结果填到 txtResult 文本框中。

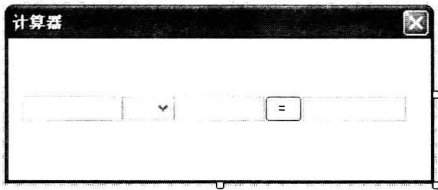


图 8-69 窗体设计

若在设计器中用鼠标双击“=”按钮，则系统会打开 C# 源代码文件“frmCalculate.cs”，然后自动插入以下代码：

```
private void btnCalculate_Click(object sender, EventArgs e)
{
}
```

可以在这个现成的方法中添加功能代码，具体代码如下：

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    try
    {
        double number1 = Convert.ToDouble(txtNumber1.Text);
        double number2 = Convert.ToDouble(txtNumber2.Text);
        double result = 0;
        switch (cboOperator.Text)
        {
            case "+":
                result = number1 + number2;
                break;
            case "-":
                result = number1 - number2;
                break;
            case "*":
                result = number1 * number2;
                break;
            case "/":
                result = number1 / number2;
                break;
        }
        txtResult.Text = result.ToString();
    }
    catch (Exception ext)
    {
        MessageBox.Show(ext.Message);
    }
}
```

在这段代码中，程序首先将两个文本框的文本转换为两个双精度浮点数，然后根据运算符下拉列表的内容，使用一个 `switch` 结构化条件语句进行数学运算，最后将运算结果填到输出文本框中。

在这段程序中以下三行代码有可能出现错误：

```
double number1 = Convert.ToDouble(txtNumber1.Text);
double number2 = Convert.ToDouble(txtNumber2.Text);
result = number1 / number2;
```

用户输入的不是数字文本，或者当进行除法运算时输入的除数为 0，这些情况都会导致程序出错。因此此处使用了“`try -- catch`”结构来捕获异常，并显示一个消息框进行提示，如图 8-70 所示。

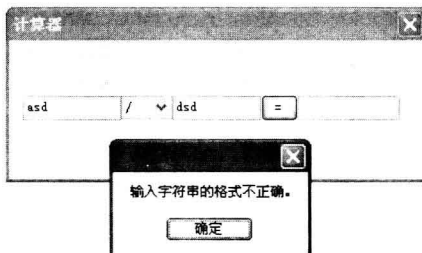


图 8-70 消息框

因为是在这里捕获的异常，所以应由程序自己来处理，这不会导致比较大的程序错误，用户关闭这个提示消息框后，程序照常运行。

- DialogResult 属性

窗体的 DialogResult 属性是一个很重要的属性，在介绍 DialogResult 属性前需要说明窗体的两种显示模式，即模式窗体和非模式窗体。

模式窗体就是对话框，也就是说，在一个应用程序中，一个对话框窗体霸占着用户的输入，用户在关闭对话框之前对应用程序的其他窗口不能进行任何操作。对话框窗体用于输入某些重要的数据，比如登录对话框用于输入用户名和密码。在 C# 中可以调用窗体的 ShowDialog 方法来以模式的方式显示一个窗体，这个方法是同步的，调用方会一直停滞在 ShowDialog 方法处，一直等到对话框被关闭后调用方才会继续运行。

非模式窗体就是正常的窗体，这个窗体显示时，用户既可以操作这个窗体，也可以操作应用程序的其他窗口。在 C# 中可以调用窗体的 Show 方法来以非模式的方式显示一个窗体，这个方法是异步的，调用方执行完 Show 方法后接着往下执行而不必等待窗体的关闭。

窗体的 ShowDialog 方法没有参数，返回值类型是“System.Windows.Forms.DialogResult”，这个类型就是用户在窗体中的操作结果状态。在对话框窗体内部，程序可以设置窗体的 DialogResult 属性来设置 ShowDialog 方法的返回值。

“System.Windows.Forms.DialogResult”类型是枚举类型，其可选值如表 8-12 所示。

表 8-12 属性值

属 性 值	说 明
None	无状态
OK	用户确认的状态，一般为用户单击了“确定”按钮
Cancel	用户取消的状态，一般为用户单击了“取消”按钮
Abort	用户忽略的状态，一般为用户单击了“忽略”按钮
Retry	用户重试的状态，一般为用户单击了“重试”按钮
Ignore	用户忽略的状态，一般为用户单击了“忽略”按钮
Yes	用户选择是的状态，一般为用户单击了“是”按钮
No	用户选择否的状态，一般为用户单击了“否”按钮

在 Windows 应用程序中，窗体对象之间可以使用公开字段和属性的方式来交换数据。对于对话框，使用 ShowDialog 方法的返回值和 DialogResult 属性是主调方和对话框之间的最重要的数据交流方式，原则上不能抛弃这个方法的返回值而完全使用其他方式。

需要注意，用户单击窗体最右上角的“关闭”按钮或调用窗体的 Close 方法只是隐藏窗体，窗体实例还是有效的，相关资源没有被释放，窗体还能再次显示。因此为了彻底关闭窗口释放资源，需要调用窗体的 Dispose 方法，在实践中经常使用 using 语法块来使用窗体对象，例如以下的代码：

```
using( Form1 dlg = new Form1( ))
{
```

```
        dlg.ShowDialog();  
    }
```

8.12.1 读写系统配置

若希望程序启动时，其用户界面的初始化的文本就是上一次运行程序时用户输入的文本，则需要程序能将用户输入的数据保存到系统配置中。

在 VS.NET 中添加系统配置是非常容易的。在解决方案资源管理器中选中“第一个 Windows 应用程序”节点，单击解决方案资源管理器工具条上的“属性”按钮，就可以显示出该 C# 工程的属性页面。

C# 工程属性页面有很多选项卡，选中“设置”选项卡，此时可以看到如图 8-71 所示的用户界面。

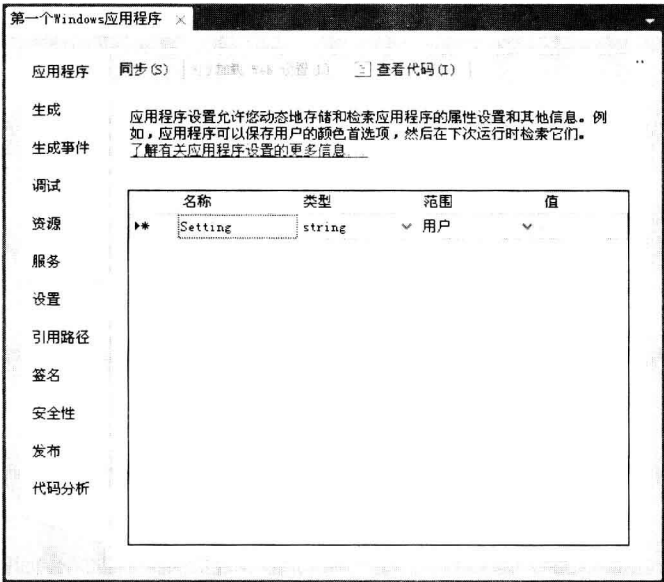


图 8-71 用户界面

在系统配置列表中，“名称”字段是设置的名称，“类型”字段指明了设置项目的数据类型，“范围”字段指明了该配置项目的作用范围，“值”字段指明了该配置项目的默认值。

需要说明的是，“范围”字段有“用户”和“应用程序”两个选项，若选择“用户”，则程序自身能修改设置信息；若选择“应用程序”，则该设置信息对程序来说是只读的，只能由用户使用另外的文本编辑器来修改配置文件。

在该用户界面中操作起来很方便，开发者可用它来向应用程序添加系统配置。如图 8-72 所示为向应用程序添加了名为“Number1”、“Number2”、“Operator”三个系统配置选项，分别对应 txtNumber1、txtNumber2、cboOperator 控件的文本。

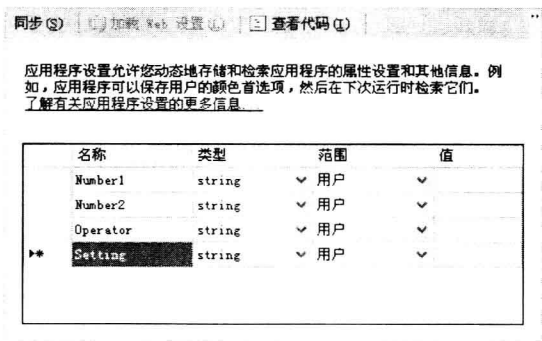


图 8-72 添加系统配置选项

切换到窗体 frmCalculate 的设计界面，双击窗体的空白区域，此时 VS.NET 会打开窗体的 C# 代码文本，然后自动插入如下代码：

```
private void frmCalculate_Load(object sender, EventArgs e)
{
}
```

这段代码是处理窗体加载事件的，窗体加载事件是指窗体对象实例化以后，准备显示前一刻的事件，可以方便开发人员对窗体进行初始化。

在这个方法中添加以下代码：

```
Try
{
    txtNumber1.Text = Properties.Settings.Default.Number1;
    txtNumber2.Text = Properties.Settings.Default.Number2;
    cboOperator.Text = Properties.Settings.Default.Operator;
}
catch
{
}
```



图 8-73 属性编辑器

在这段代码中，将三个系统配置值填充到用户界面上的控件中，由于这个过程容易出错，因此进行了错误处理。由于根据系统配置初始化用户界面并不是关键功能，因此无须向用户提示，发生错误也没什么事，无所谓。

当程序退出时，程序还需要将当前用户界面的数值保存到系统配置中。打开窗体的设计界面，然后按下属性编辑器工具条上的“事件”按钮，此时属性编辑器用户界面如图 8-73 所示。

双击 FormClosing 项目，此时 VS.NET 会打开窗体的 C# 代码文件，并自动插入以下源代码：

```
private void frmCalculate_FormClosing(object sender, FormClosingEventArgs e)
{
}
}
```

这段代码能处理窗体关闭时的事件，在这个方法中添加以下代码：

```
Properties.Settings.Default.Number1 = txtNumber1.Text;
Properties.Settings.Default.Number2 = txtNumber2.Text;
Properties.Settings.Default.Operator = cboOperator.Text;
Properties.Settings.Default.Save();
```

这段代码是将控件中的数据设置到系统配置对象的属性中，然后调用系统配置对象的 Save 方法将数据保存到磁盘上的文件中。这样当下次程序启动时会调用窗体的“frmCalculate_Load”方法，这个方法会从磁盘中加载系统配置数据并设置用户界面控件的数据。

8.13 调试

程序是复杂的，不可能把代码写出来后就能保证完全正确，可能存在编译错误和运行时错误。

编译错误就是程序源代码写得不对，不符合 C#的基本语法。排除编译错误很简单，就是根据编译器给出的提示信息修改源代码，使其符合 C#语法。

运行时错误就是程序能编译成功，但在启动运行后发生了错误。这种错误包括功能不对、系统抛出未处理的异常等。排除运行时错误就比较难了。

VS.NET 为排除运行时错误提供了强大的调试功能支持，与其他开发环境类似，提供了设置断点、单步调试。

8.13.1 执行代码

VS.NET 提供了如图 8-74 所示的“调试”工具条。






图 8-74 “调试”工具条

在该工具条中，常用的按钮如表 8-13 所示。

表 8-13 常用按钮及功能

功 能 按 钮	快 捷 键	说 明
启动调试/继续调试	F5	能以调试模式启动应用程序
全部中断	Ctrl+Alt+Break	中断当前应用程序的执行，停止在当前运行的代码处，并显示相关的代码文件
停止调试	Shift+F5	强制立即关闭当前以调试模式运行的应用程序，这可能会导致数据丢失
重新启动	Ctrl+Shift+F5	强制立即关闭当前以调试模式运行的应用程序，然后重新编译程序，再次以调试模式启动应用程序

续表

功 能 按 钮	快 捷 键	说 明
 逐语句	F11	以调试模式启动应用程序，然后一条语句一条语句地执行程序。若当前语句调用了 一个方法，则还会进入该方法体内部进行单步调试运行。VS.NET 会打开相应的源代码文件并突出显示当前执行的代码文本
 逐过程	F10	以调试模式启动应用程序，然后一条语句一条语句地执行程序。若当前语句调用了 一个方法，则不会进入方法体内部继续单步调试运行，而是等待方法执行完毕 后接着单步调试运行
 跳出	Shift+F11	在断点或单步调试运行程序的过程中，退出单步调试模式，全速运行程序。当本方法的代码运行完毕时，当前执行位置跳出了本方法，然后再以单步调试模式运行程序

通过这些功能按钮，开发者就能调试程序，帮助发现程序中的逻辑错误。

当程序很复杂时，若全部单步调试还是非常烦琐的，此时开发者可以使用断点来让程序快速运行到指定位置。在 VS.NET 的代码窗体中，可以在可能导致逻辑错误的源代码那行的前面，用鼠标左击一下即可在这行代码中插入一个断点。插入断点后其代码窗体的界面如图 8-75 所示。

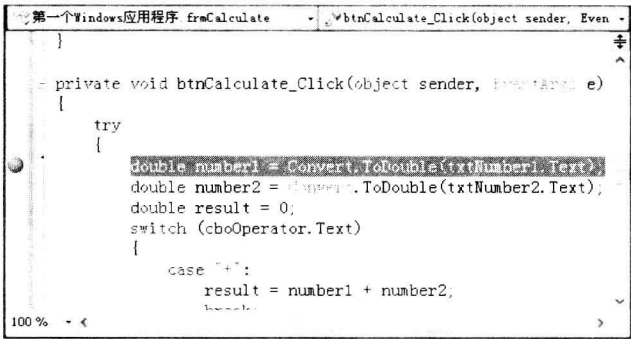


图 8-75 代码窗体

按下 F5 键以调试模式运行程序。当单击用户界面上的“=”按钮时，代码执行就遇到了上面设置的窗体，VS.NET 会立即暂停程序的运行，并突出显示正在运行的代码。此时代码窗体用户界面如图 8-76 所示。

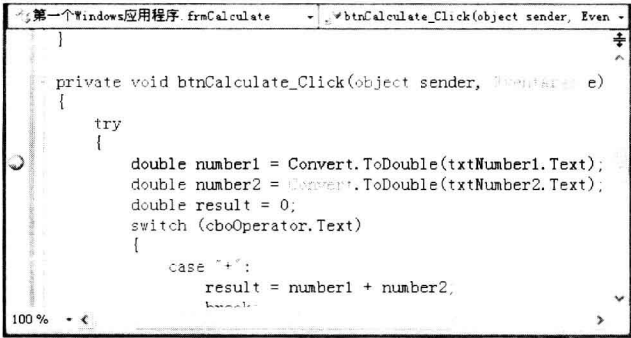


图 8-76 代码窗体

在代码窗体中，以黄色背景突出显示了当前代码。按一下 F11 键进行一次单步运行，则代码运行到下一条语句，此时代码窗体用户界面如图 8-77 所示。

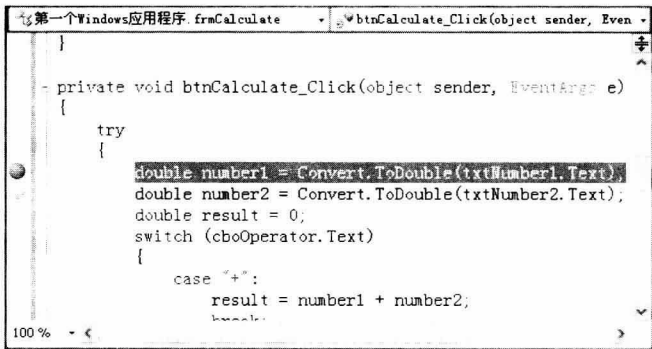


图 8-77 代码窗体

用户可以用鼠标拖曳左边那个表示当前代码位置的黄色向右箭头标识，来选择新的当前代码执行位置。这样可以临时改变代码逻辑，跳过或重复执行某些代码。

用户还可以修改源代码文本，修改完后再接着进行调试。在大多数情况下，用户修改源代码文本时无须中断程序，VS.NET 会重新编译并将新代码掺到旧代码中投入使用，即参与调试和运行，新代码中还可以设置断点，这点与 VB 很类似。

8.13.2 查看和修改变量值

VS.NET 中提供一个“局部变量”的工具窗格，能显示出当前值的代码处所能访问的所有局部变量的名称和数值，该窗格用户界面如图 8-78 所示。

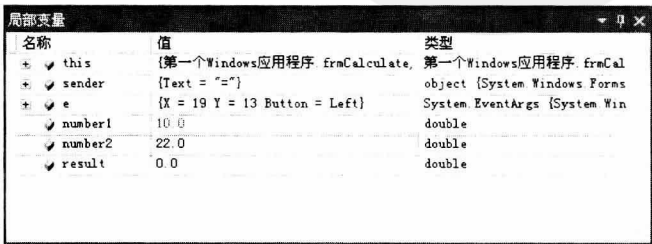


图 8-78 “局部变量”窗格

“局部变量”窗格包含一个列表，该列表列出了局部变量的名称、数值和数据类型。若单步执行某段代码而导致变量值发生了改变，则变量值也会同步更新，并以红色突出显示变量值。

例如，在执行代码“double number1 = Convert.ToDouble(txtNumber1.Text);”前，变量 number1 显示为 。执行了这行代码后，变量 number1 显示为 。

在该列表中，用户可以直接修改变量值，比如在此列表中用鼠标双击“number1”后面的数值“0.0”，则该单元格处于可编辑的状态，此时就可以输入新的变量值了，如图 8-79 所示。

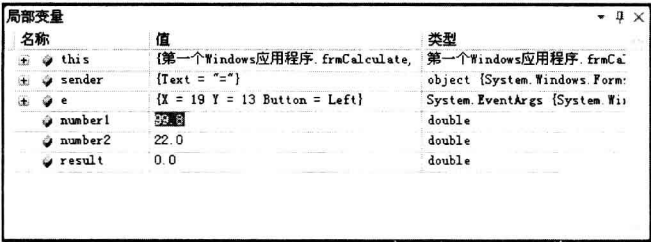


图 8-79 修改变量值

VS.NET 还提供“监视”窗格用来显示指定变量，其界面如图 8-80 所示。

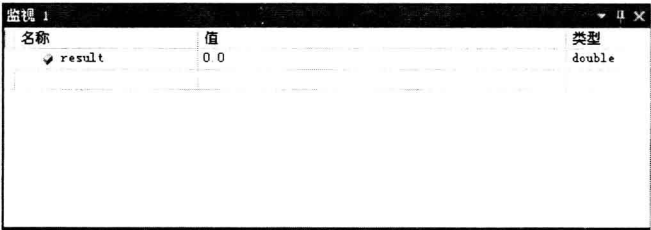


图 8-80 “监视”窗格

“监视”窗格和“局部变量”窗格内容和功能一样，只是“局部变量”窗格中的变量是根据当前执行语句的上下文自动列出来的，而“监视”窗格中的变量是可以增加和删除的。

往“监视”窗格中添加变量很简单，在源代码中选中变量名，然后将鼠标拖曳到“监视”窗格中即可。在“监视”窗格中按下 Delete 键，或者使用快捷菜单中的“删除监视”菜单项目即可删除选中的正在监视的变量。

在 VS.NET 中，除了在“局部变量”和“监视”窗格中查看和修改变量值外，开发者还可以将鼠标移动到某个变量名之上，然后系统会弹出一个小的提示信息，列出变量名和数值。例如当鼠标移动到代码“result”之上时，如图 8-81 所示，系统会显示出一个小的提示信息。

鼠标移动到这个提示信息上，单击变量数值，此时用户也可以修改变量值。此时的用户界面如图 8-82 所示。

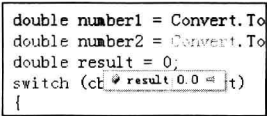


图 8-81 提示信息

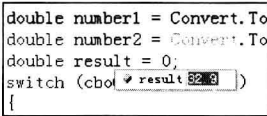


图 8-82 用户界面

在用户界面弹出的小文本框中，用户可以输入新的变量值，然后按下 Enter 键确认操作即可修改变量值。

8.13.3 命令窗口

在调试中，“命令窗口”也是很有用的，它是一个调试过程中的命令行窗口。其用户界面如图 8-83 所示。

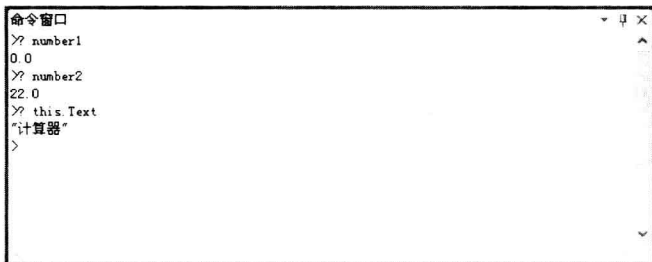


图 8-83 命令窗口

这个“命令窗口”和 VB 中的“立即窗口”很类似，只是语法有些不同。它采用 C#样式的表达式，包括逻辑表达式和数学表达式，还能执行对象的属性和方法。

在“命令窗口”中，输入命令“>? 变量名”按下 Enter 键即可查看变量值。例如，输入命令“>? number1”按 Enter 键即可显示变量“number1”的数值。

输入命令“>? 变量名= 数值”即可修改变量值。例如，输入“>? number1=87.3”即可修改变量“number1”的值为“87.3”。

输入命令“>? this.Text”即可查看当前窗体的标题。

8.14 测试和运行 Windows 应用程序

当上述所有的功能都完成后，这个 Windows 应用程序就开发完毕，可以测试和运行了。

单击 VS.NET 的主菜单项目“调试→启动调试”，或者按下工具条上的“启动调试”按钮，或者按下快捷键 F5，都可以编译运行这个 Windows 应用程序。该程序运行时的用户界面如图 8-84 所示。

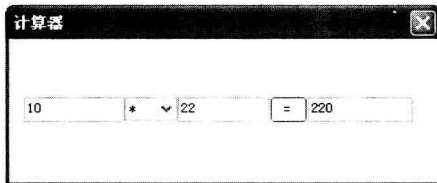


图 8-84 用户界面

在 C#项目编译成功后，会在工程所在目录下的“bin\Debug”子目录下创建一个名为“第一

个 Windows 应用程序.exe”的可执行文件。这个文件就是程序文件。

8.15 小结

在本章用 VS.NET 2010 开发了一个简单的 Windows 应用程序。介绍了 Windows 窗体设计的一些原则，Windows 应用程序的用户界面驱动的原理，此外还介绍了 VS.NET 提供的代码调试功能。

第 9 章

开发第一个 ASP.NET 应用程序

在本章将开发第一个 ASP.NET 应用程序。仿造第 8 章中的“第一个 Windows 应用程序”的功能，也做一个用于进行加、减、乘、除计算的 ASP.NET 应用程序。

9.1 ASP.NET 概念

通俗地说，ASP.NET 应用程序是使用 C# 开发的 B/S 程序，所谓 B/S 程序就是采用 Browser/Server 架构的应用程序，俗称网页程序、网站程序。这里的 Browser 就是网页浏览器，能解析 HTML 代码以文本、图片或多媒体的方式展现文档；Server 就是 Web 服务器，是产生这些 HTML 文档内容的服务器程序，它只产生文档，不显示文档，通过网络发送到网页浏览器再显示出来，这个网络可以是局域网或因特网。

9.1.1 B/S 架构

B/S 就是 Browser/Server 架构，是目前使用很多的应用系统架构，特别适用于因特网应用系统。它最大的优点就是部署和维护起来很方便，而且用户界面能做得比较美观华丽。

图 9-1 是 B/S 架构原理图。



图 9-1 B/S 架构原理图

一句话描述 B/S 架构就是：服务器生成 HTML 文档，以 HTTP 协议通过计算机网络传输到客户端计算机中，被网页浏览器显示出来供用户查看。

B/S 系统架构是以 HTTP 传输协议和 HTML 文档规范为基础的。B/S 架构的系统运行过程如图 9-2 所示。

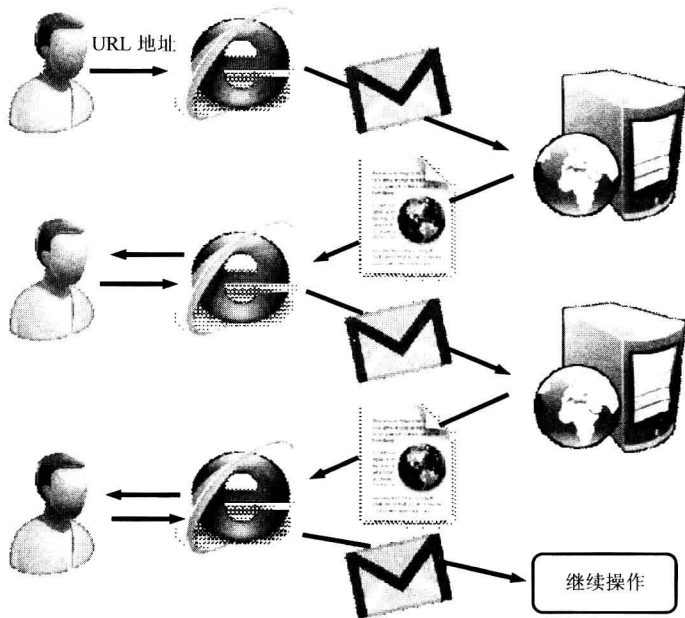


图 9-2 B/S 架构的系统运行过程

在这个过程中需要执行以下步骤：

- (1) 用户在客户端浏览器输入 URL 地址开始使用系统。客户端浏览器通过 URL 地址访问 Web 服务器。
- (2) Web 服务器收到客户端浏览器的请求，执行某些操作，然后根据操作结果生成 HTML 文档，并通过网络发送到客户端。
- (3) 客户端浏览器收到服务器发送的 HTML 文档并显示出来。
- (4) 用户看到浏览器中显示出来的 HTML 文档，做一些命令操作，比如想要删除某条数据。
- (5) 客户端浏览器接收用户的操作，将其转换为某种内部命令的格式，然后生成数据包，并通过网络发送给 Web 服务器。
- (6) Web 服务器接收客户端发送的数据包，解析其中的命令，执行用户指定的操作，比如删除某条数据。然后根据执行结果重新生成 HTML 文档，并通过网络发送给客户端浏览器。
- (7) 客户端浏览器收到服务器端发送的 HTML 文档并显示出来。

(8) 用户看到浏览器中显示出来的 HTML 文档, 做一些命令操作, 比如想要新增一条数据。

(9) 客户端浏览器接收用户的操作, 将其转换为某种内部命令的格式, 然后生成数据包, 并通过网络发送给 Web 服务器, 如此循环, 可以到永远。

(10) 用户通过在系统中的注销操作, 或者直接关闭浏览器程序, 将客户端退出系统。但此时 Web 服务器还在运行, 还可以和其他的客户端打交道。

B/S 系统之所以现在非常流行, 是因为其具有巨大优势, 主要有:

(1) 部署简单。B/S 系统是一个瘦客户端系统。客户端只需要安装网页浏览器即可使用系统, 而无须像 Windows 应用系统那样要装客户端软件, 因此没有客户端软件部署和维护的工作, 只要能上网就能使用系统。

(2) 维护简单。B/S 系统的功能都集中在一台或少数几台 Web 服务器上, 因此可以进行软件的集中部署和维护, 可实现远程维护, 工作量小, 成本低。

(3) 扩展性好。B/S 的功能模块是基于页面文件的, 开发人员只需要新增页面文件即可添加系统功能, 扩展性好。

(4) 用户界面华丽。使用 HTML 丰富的用户界面功能, B/S 系统的用户界面可以做得比较华丽漂亮。

当然 B/S 也有其缺点, 主要有:

(1) B/S 系统必须在线运行, 若客户端不能连接到 Web 服务器就完全不能使用 B/S 系统, 没有离线运行的功能。相比而言, C/S 系统能实现一定的离线运行能力。

(2) B/S 用户体验不够好, 很依赖鼠标操作, 数据录入速度慢, 很多用户界面操作效果难以实现或不能实现, 很多领域 B/S 系统无法参与。

(3) 对开发者来说, B/S 系统是无状态的, 业务数据存储和页面数据交流比较困难, 这个问题比较麻烦。

9.1.2 HTTP 传输协议

HTTP (Hyper Text Transfer Protocol) 传输协议是 B/S 系统的基础, 是 B/S 客户端和服务端两者交流使用的数据传输协议, 它是建立在 TCP/IP 底层协议之上的。

根据 HTTP 协议, 客户端和服务端传输数据的步骤如下:

(1) 客户端浏览器使用一个 URL 地址, 通过网络上的 DNS 服务器解析出服务器 IP 地址。比如对于 URL 地址 “http://www.cnblogs.com/xdesigner/archive/2010/03/15/1686165.html”, 将 URL 拆分成 “http://www.cnblogs.com” 和 “xdesigner/archive/2010/03/15/1686165.html” 两部分。对于 “http://www.cnblogs.com” 通过 DNS 解析知道服务器 “cnblogs.com” 的 IP 地址为 61.135.169.116。然后浏览器和这个 IP 地址的服务器通过 TCP/IP 协议建立起稳定的网络连接,

服务器端默认采用 80 端口进行 TCP/IP 请求监听。

(2) 客户端浏览器将 HTTP 请求数据发送到服务器。比如将 URL 拆分得到“xdesigner/archive/2010/03/15/1686165.html”地址信息，还有客户端的表单数据，相关的系统配置信息等。

(3) 服务器端接收客户端浏览器发送过来的数据，解析并执行一些操作，然后生成一些数据通过网络回发给客户端浏览器。比如服务器收到“xdesigner/archive/2010/03/15/1686165.html”，然后分析这段指令，生成 HTML 文档，再回发给客户端浏览器。

(4) 客户端浏览器收到服务器发送的 HTML 文档后，断开 TCP/IP 连接，然后解析并显示这个 HTML 文档。

从这里看出 HTTP 传输协议就是客户端和服务端之间的一问一答。比如上例中，客户端向名为“www.cnblogs.com”的服务器问：“xdesigner/archive/2010/03/15/1686165.html 的内容是什么啊？”；服务器回答：“好的，这是文件 xdesigner/archive/2010/03/15/1686165.html 的内容……”

在计算机网络中，一台服务器能同时应付多个客户端，形成如图 9-3 所示的结构。

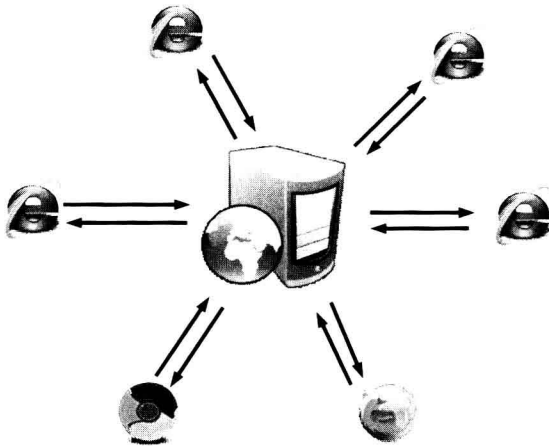


图 9-3 网络结构

在 HTTP 传输协议中，客户端是躲在暗处的，IP 是变化的；而服务器是处在明处的，IP 是固定的，因此只能是客户端询问服务器，而不会是服务器主动向客户端推送数据。客户端不限于浏览器程序，可以是其他应用程序，服务器不会限制客户端的类型，只要大家能遵守 HTTP 传输协议即可。

由于 HTTP 传输协议一问一答完成后就关闭了，因此造成一种数据无状态的情况。服务器在某次 HTTP 交流中生成的保存在内存中的数据会因为网络连接的断开而完全消失，这样服务器对每一次连接都认为对方是一个全新的客户端。

比如某 B/S 系统需要用户验证进行登录。用户张三在客户端向服务器发送了用户名和密码进行用户登录操作；服务器验证了用户名和密码，认可张三的身份，回发的 HTML 文档便说明认

可张三的身份。若此时网络连接立即断开，则保存在服务器内存中的对张三的验证信息会被清除，此时张三再次访问服务器，服务器就认为他是一个新的使用者，要求提供用户名和密码进行验证，这个 B/S 系统就永远提示用户进行登录，也就没法用了。

为此 B/S 系统将客户端浏览器 Cookie 技术和服务器端 Session 技术一起配合使用来解决这个问题，如图 9-4 所示。

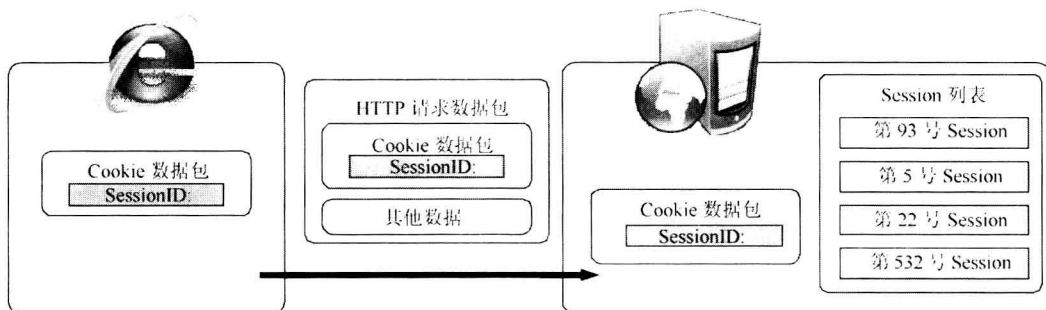


图 9-4 两种技术配合使用

浏览器在客户端保存一个名为 Cookie 的小数据包，浏览器是不会修改这个数据包的。与此同时服务器端维护一个 Session 列表，这个列表是全局的，只要服务器运行着，该列表就一直存在。该列表中保存着若干个 Session 对象，每个 Session 对象都有一个唯一的编号，Session 对象是一个容器，应用程序可以以字典的方式在其中存储数据。

浏览器每次向服务器发送请求时都会将 Cookie 数据发送到服务器端，服务器端接收到客户端发来的请求数据后会识别其中的 Cookie 数据，挑出其中的用做 SessionID 的数据，然后在 Session 列表中查找指定编号的 Session 对象。

若服务器没有找到指定编号的 Session 对象，则所出现的情况如图 9-5 所示。

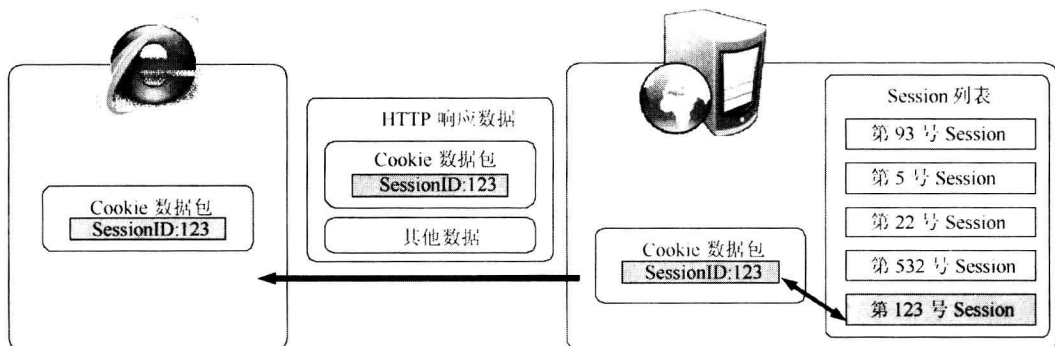


图 9-5 服务器没有找到指定对象的情况

此时服务器就会在 Session 列表中创建一个 Session 对象，分配一个编号，并将新的 SessionID 号填到 Cookie 数据包中。然后将修改后的 Cookie 数据包含在 HTTP 响应数据包中回

发给客户端浏览器。

客户端浏览器接收 HTTP 响应数据后断开网络，然后解析出新的 Cookie 数据并保存好。

客户端浏览器再次向服务器发出请求，如图 9-6 所示。

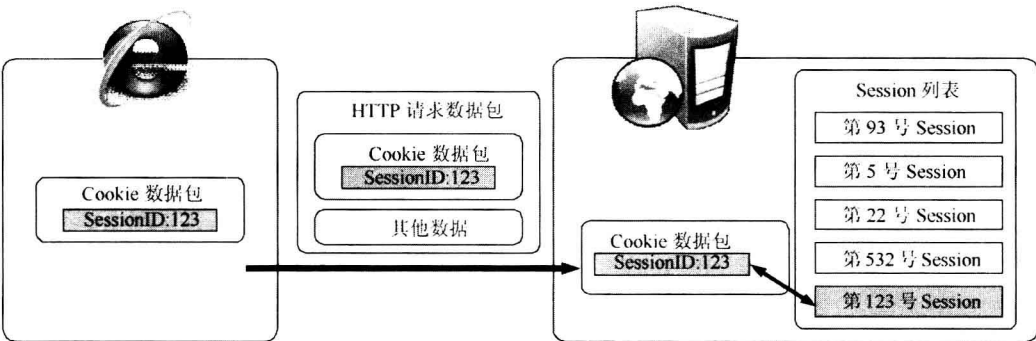


图 9-6 客户端浏览器再次向服务器发出请求

此时客户端浏览器仍然会将 Cookie 数据包含在 HTTP 请求数据包中，而此时的 Cookie 中通过上一次与服务器的交流已经包含了一个有效的 SessionID 数据。

服务器接收 HTTP 请求数据，解析 Cookie 数据获得 SessionID 值，然后在 Session 列表中查找指定编号的 Session 对象，结果找到了指定的 Session 对象，于是将其作为当前 Session 对象。服务器端应用程序可以往这个 Session 对象读取或保存数据，实现了在无状态的 HTTP 传输协议上的有状态，从而将多个相互之间毫无关系的 HTTP 问答联系在一起，形成一个有机的整体。

比如应用程序将用户验证信息存储在 Session 中，则用户第一次使用系统是在验证成功后，以后使用系统，应用程序就可以在 Session 中获得验证信息而无须用户再次验证了。

这种客户端 Cookie 和服务端 Session 结合的技术是以客户端 Cookie 中的 SessionID 数据为基础的，因此存在安全漏洞，如图 9-7 所示。

黑客会窃取合法用户正在使用的浏览器的 Cookie 中的 SessionID 值，并以此冒充合法用户来连接服务器，由于服务器只认 SessionID 不认其他的，因此系统将无条件地服从黑客的命令。实际上前几年这种黑客攻击手段在网上很流行，造成不少网络安全事故。

针对这种安全漏洞，可以采用两种技术手段来解决这个问题。

一种是提醒用户使用系统提供的注销功能退出系统，而不是简单地直接关闭浏览器。注销功能能删除 Cookie 中保存的 SessionID，并在全局 Session 列表中删除相应的 Session 数据，这样能让 SessionID 无效，成为永久的历史。任何人都无法使用无效的 SessionID 来使用系统。

另一种就是服务器端 Session 超时。比如设置服务器 Session 超时时间为 20 分钟，若这 20 分钟之内客户端浏览器没有和服务器交流，则服务器会自动地将 Session 删除，而 SessionID 自然也就无效了。此时即使登录过的客户端连接服务器，系统也会要求进行身份验证。这种方式能

缩短可能被黑客攻击的时间，提高安全性。

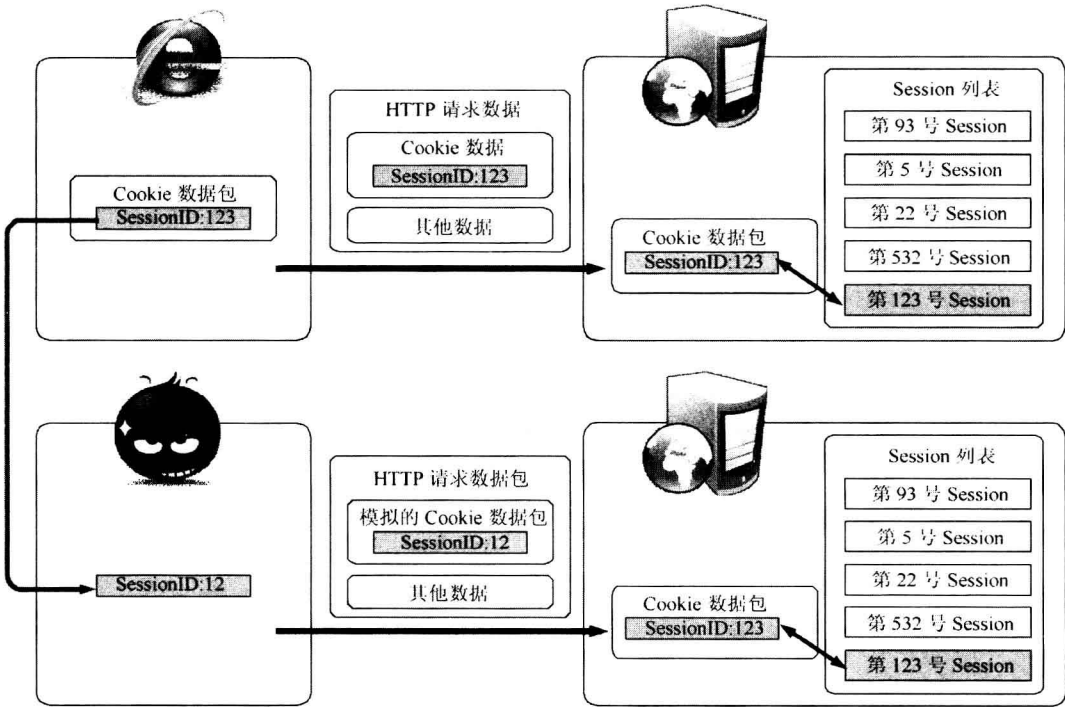


图 9-7 存在安全漏洞

9.1.3 HTML 文档规范

B/S 系统中服务器端交付给客户端的数据绝大多数都是 HTML 文档。对于开发者来说，HTML 文档就是符合 HTML 文档规范的字符串。对于 B/S 程序员，不管是用 ASP.NET 还是 Java 等，都必须理解 HTML 文档规范。

HTML (Hyper Text Markup Language)，即超文本标记语言，是用于描述网页文档的一种标记语言。

从使用者的角度看，它具有以下三大功能：

- (1) 丰富的数据展示方式。HTML 文档是超媒体文档，它能包含文字、图像、声音、视频等多媒体形式，以丰富多彩的方式展示各种数据。
- (2) 超链接。HTML 文档中可以包含若干个超链接区域，用户单击超链接区域就能显示出其他的 HTML 文档，这种方式无论其定义和使用都是非常简单的，能实现多个网页之间的互联互通。
- (3) 动态内容。在 HTML 文档的基础上能实现 DHTML 技术，使用客户端脚本技术来实现

各种动态内容效果，能提供很不错的用户体验。

从开发者的角度看，HTML 具有以下三大特点：

(1) HTML 文件内容是纯文本的，其语法比较简单，其内容阅读和生成都比较容易。

(2) HTML 文档具有松散的结构，由于 HTML 文件本身是纯文本的，不能包含其他的图片、声音、视频等媒体数据内容，只能引用它们的数据文件，因此增加了开发难度。

(3) HTML 是国际标准，任何 B/S 程序，包括 ASP.NET、Java 或其他技术生成的 HTML 文档都是没有差别的。

以下是一个 HTML 文档内容的范例。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>这是 HTML 标题</title>
  </head>
  <body>
    <font color="red" >这是 HTML 文档的内容。</font>
    <a href="http://www.cnblogs.com/">这是一个超链接</a>
  </body>
</html>
```

从这个范例可以看出，一个 HTML 文档主要包括以下几个部分：

(1) 文档开头具有“<!DOCTYPE>”元素，用于说明该 HTML 文档采用什么规范。

(2) HTML 文档只有一个根节点，就是文本标记“<html> </html>”表示的节点。

(3) 根节点下面有 head 节点和 body 节点，head 节点用于容纳文档标题等信息，body 节点用于容纳文档内容。

对于 HTML 文档的内容可以用两种方式来理解。

(1) 将 HTML 文档内容当做字符串来理解。HTML 文档就是一个字符串，输出 HTML 文档就是输出一个字符串。在服务器端生成 HTML 文档时对 HTML 文档可以采用这种理解方式。

(2) 将 HTML 文档内容当做一个树状的数据结构来理解。有一个类型为“html”的根节点，根节点下面有若干个子节点，子节点下面还有子节点。在客户端处理 HTML 文档时可以采用这种理解方式。因为客户端处理 HTML 文档基本上就是用 JavaScript 开发的，而 JavaScript 技术的基础就是 HTML DOM，而 DOM 就是以树状结构来理解文档的。

9.1.4 ASP.NET 服务器端架构

ASP.NET 是微软原先提出的 ASP 技术的升级版本，图 9-8 是 ASP.NET 应用系统的架构。

Web 服务器是 Web 应用系统中的核心部分，它逻辑上是一台服务器对象，实际上可以由一台服务器或多台服务器集群而成。Web 服务器承载着系统大部分的存储和运算任务，是 Web 应用系统中最为繁忙的部分，Web 服务器包含以下内容。

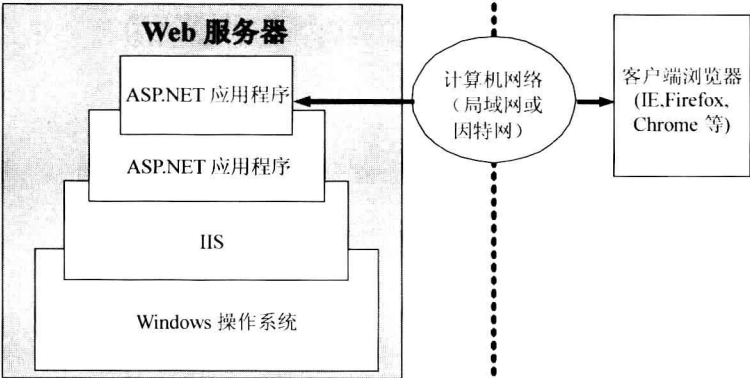


图 9-8 ASP.NET 应用系统的架构

• Windows 操作系统

在这个架构中，Windows 操作系统是最底层的部分，这个 Windows 版本是 Windows 2000/XP 或更高版本，推荐的是 Windows Server 2003/2008 版本。

• IIS

IIS 就是微软的 Internet Information Server 的简称，它是 Windows 的一个可选组件，它主要包含 IIS Admin(IISADMIN)和 World Wide Web Publishing(W3SVC)两个 Windows 服务，而且 W3SVC 服务又依赖于 IISADMIN 服务。

打开 Windows 控制面板，再打开管理工具，然后运行 Windows 服务管理器，该管理器的用户界面如图 9-9 所示，在图中就能看到这两个服务的信息。图 9-10 显示的是 IIS Admin 的属性。

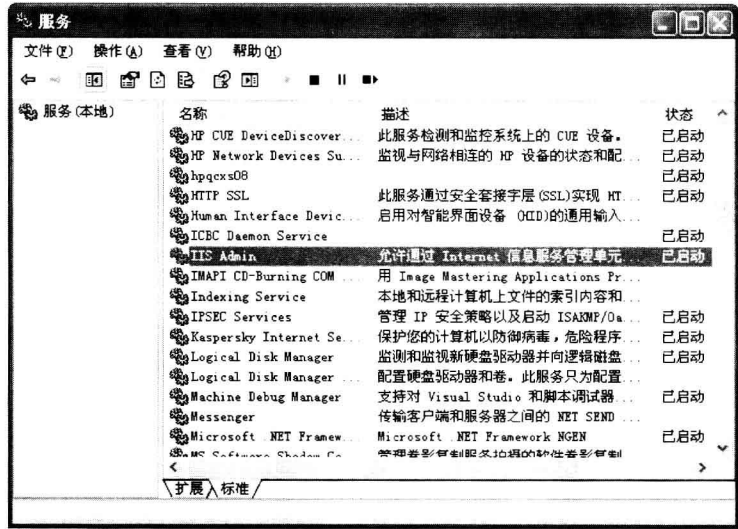


图 9-9 Windows 服务管理器

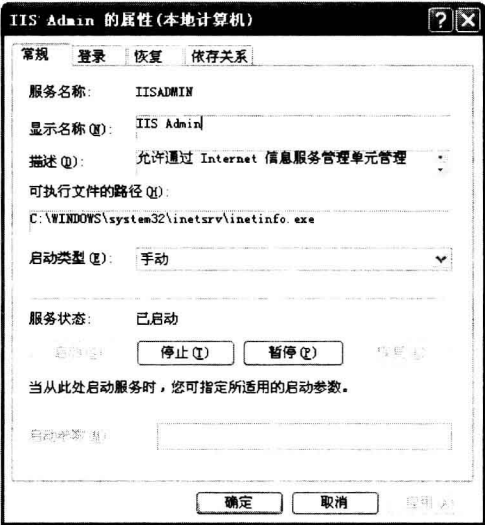


图 9-10 IIS Admin 的属性

图 9-11 显示的是 World Wide Web Publishing 的属性。

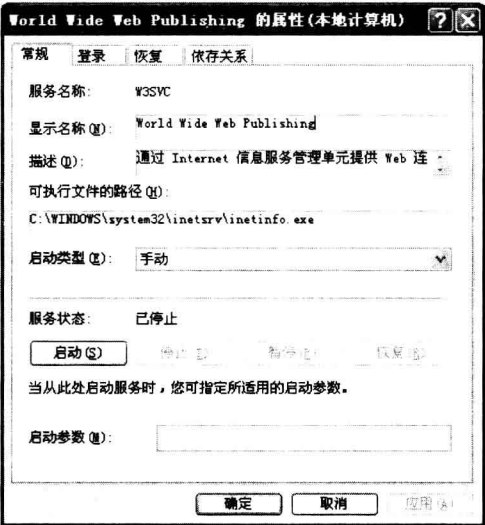


图 9-11 World Wide Web Publishing 的属性

IIS 是微软 Web 技术的基础，ASP 和 ASP.NET 的运行都是以它为基础的。开发者可以使用它的 Internet 信息服务管理器来管理网站。开发者在 Windows 控制面板的管理工具中运行该管理器，其用户界面如图 9-12 所示。

开发者可以使用这个管理器来启动或停止网站，设置虚拟目录和进行安全配置等操作。

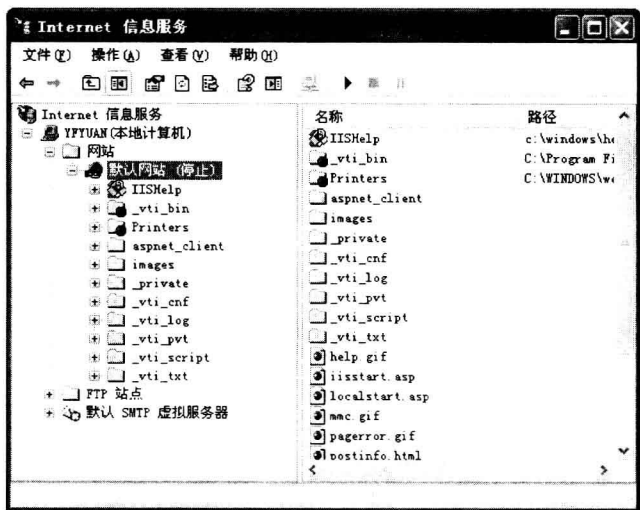


图 9-12 Internet 信息服务

- ASP.NET 应用程序框架

ASP.NET 应用程序框架是微软.NET 框架中的一部分。它运行在 IIS 之上，能编译执行 ASP.NET 应用程序，提供一些诸如缓存、安全控制等服务器控制，还提供了丰富的供 Web 使用的类库和控件库。

- ASP.NET 应用程序

ASP.NET 应用程序就是开发者针对各种应用开发的基于 ASP.NET 技术的应用程序，它运行在 ASP.NET 应用程序框架之上。

ASP.NET 应用程序包含 ASP.NET 页面、控件、各种资源文件和脚本代码文件，其最终功能就是为了生成 HTML 文档供客户端浏览器查看，此外还能接收客户端用户发出的命令来执行相应的功能。

如图 9-13 所示，ASP.NET 应用程序框架对应用程序提供了两种支持：ASP.NET 运行时和 ASP.NET Web 控件库。

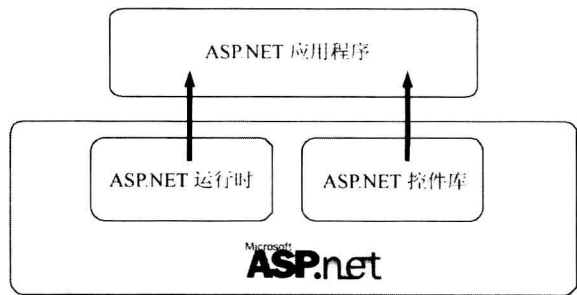


图 9-13 ASP.NET 应用程序框架

ASP.NET 运行时是后台支持，能解析执行 ASPX 页面，封装了对 HTTP 传输协议的支持，还提供了安全方面的处理。

ASP.NET 控件库对应用开发提供了很多可视化的控件，这些控件能降低 ASP.NET 的开发成本，提高开发速度并改善用户体验。

9.1.5 ASP.NET Web 服务器控件技术

ASP.NET 提供了 Web 服务器控件技术来实现所见即所得的 Web 应用程序开发工作体验，能降低开发工作量，提高开发速度。

1. HTML 文档生成技术

任何 Web 程序都是生成 HTML 文档供客户端浏览器显示的，因此任何 Web 程序的核心都是 HTML 文档技术，图 9-14 是 ASP.NET 中 HTML 文档生成技术的原理图。

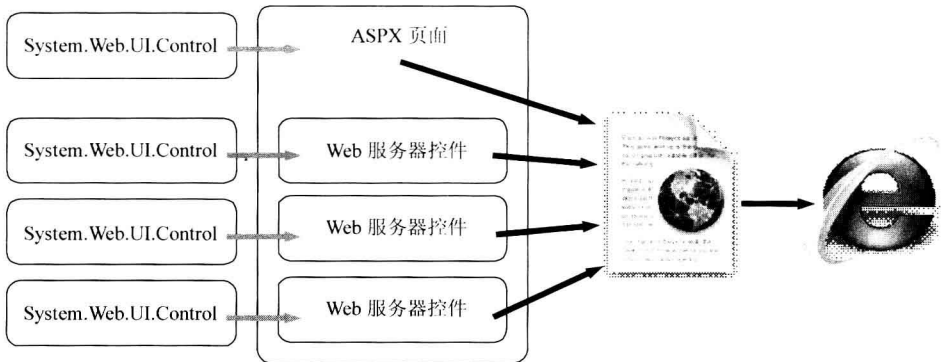


图 9-14 ASP.NET 中 HTML 文档生成技术的原理图

在 ASP.NET 开发中，开发者将 Web 页面模块化了，借鉴 WinForm 开发中控件的概念，搞出了 Web 控件。

对于每一个 ASPX 页面，可以在其中放置若干个 Web 控件，此时输出 HTML 代码不仅仅全靠 ASPX 页面，其中 Web 服务器控件也贡献了不少的 HTML 代码，然后将 ASPX 页面和 Web 控件生成的 HTML 代码集合在一起形成一个完整的 HTML 文本代码，最后发往客户端浏览器。

所有的 Web 控件以及 ASPX 页面都是从类型 System.Web.UI.Control 派生的。Control 类型定义了一个名为 RenderControl 的虚函数，这个函数就是输出展示控件内容的 HTML 代码，其派生的 Web 控件和 ASPX 页面需要重载这个函数，输出自定义的 HTML 代码。

对于某 ASPX 页面，其 HTML 代码如下：

```
<%@ Page Title="主页" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="第一个 ASPNET 应用程序._Default" %>
<html>
```



```

<body>
<h2>
    欢迎使用 ASP.NET!
</h2>
<p>
    <asp:Button ID="Button1" runat="server" Text="Button" />
</body>
</html>

```

这段代码中被方框选中的代码表示一个按钮类型的 Web 控件。ASP.NET 框架程序执行这个 ASPX 页面的步骤如下。

(1) 系统通盘解析这个 ASPX 页面的 HTML 代码，试图解析出表示 Web 服务器控件的 HTML 代码。例如遇到 “<asp:Button ID="Button1" runat="server" Text="Button" />”，发现其标签名前缀为 “asp” 而且有属性 “runat="server"”，说明这就是一个 Web 服务器控件，系统会据此创建一个类型为 System.Web.UI.WebControls.Button 的对象实例，设置其 Text 属性值为 “Button”。

(2) 第一行 HTML 代码是被 “<%@ %>” 包含的，这是页面指令，忽略掉，不输出。

(3) 从代码 “<html>” 开始原样输出 HTML 代码。

(4) 遇到 “<asp:Button ID="Button1" runat="server" Text="Button" />”，立即调用前面创建的 System.Web.UI.WebControls.Button 对象的 RenderControl 方法输出 HTML 代码，对于这个 Button 对象就会输出 HTML 代码片段 “<input type="button" id="Button1" value="Button" />”。

(5) 处理完这个按钮控件后，系统接着输出后续的 HTML 代码。

(6) 输出完 HTML 代码，系统会销毁 ASPX 页面对象实例和 Web 控件对象实例。

整个过程完成后，该 ASPX 页面输出的 HTML 代码如下：

```

<html>
<body>
<h2>
    欢迎使用 ASP.NET!
</h2>
<p>
    <input type="button" id="Button1" value="Button" />
</body>
</html>

```

这就是一个很标准很完整的 HTML 文档，然后服务器会发送这份 HTML 代码到客户端浏览器。

2. 服务器控件事件技术

ASP.NET Web 控件还提供了服务器控件事件技术，如图 9-15 所示。

在 ASPX 页面使用代码 “<asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />” 插入一个按钮控件，该按钮控件在客户端的 HTML 代码为 “<input type="submit" name="Button1" value="Button" />”。

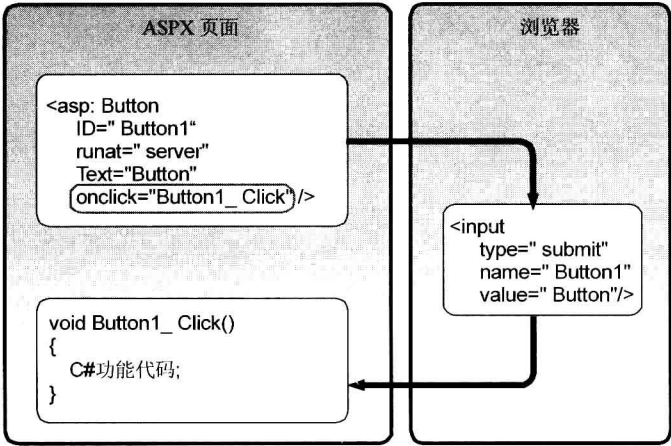


图 9-15 ASP.NET Web 控件提供的服务器控件事件技术

若客户端用户单击该按钮，则系统自动回发到服务器，服务器调用这个页面，并根据按钮名称“Button1”以及属性“onclick="Button1_Click"”来查找与页面对应的 C# 代码，找到名为“Button1_Click”的页面类型成员方法然后调用它。这样就可以很简单地将 Web 控件事件绑定到 C# 代码，实现服务器控件事件技术。

但是这种方式需要页面回发从而导致刷新，因此用户体验并不好。现在使用的 AJAX 技术就是对这种服务器控件事件的改良版。

3. 页面状态技术

ASP.NET 还实现了页面状态技术，用于解决 HTTP 传输无状态的问题。

由于 HTTP 传输过程没有状态，而系统运行中会产生大量的状态数据，因此需要在多次页面回话中使用。例如程序 C# 代码中设置了某个文本框控件的背景颜色，则系统运行过程如图 9-6 所示。

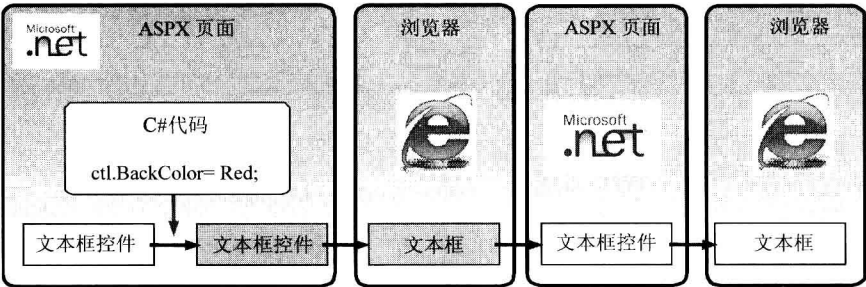


图 9-16 系统运行过程

第一步，文本框控件创建时默认为白色背景色，而第一次页面处理中有段 C# 代码修改了文本框控件的背景色为红色。

第二步，服务器端的文本框控件输出了一个 HTML 文档中文本输入框的 HTML 代码，并设置背景色为红色，结果在 HTML 文档发往客户端浏览器中，显示出一个红色背景的文本框。

第三步，浏览器发起第二次会话，将文本框中输入的文本发送到服务器，但浏览器没有发送文本框背景色信息的功能，因此服务器无法得到文本框背景色信息。

第四步，系统根据 ASPX 页面代码创建一个文本框控件，由于无法得到客户端文本框背景色，因此让文本框控件保持默认的白色背景色。然后这个白色背景的文本框控件生成了具有白色背景的 HTML 文本框代码发往客户端。

第五步，客户端浏览器接收 HTML 代码并呈现出一个白色背景色的文本框。

从这里可以看出，由于 Web 应用程序是无状态的，因此在上述的第一步中设置了文本框控件为红色背景色，但对于用户来说，文本框的红色背景只是昙花一现，执行到第三步状态就没了。为了维持状态，需要在第四步时使用 C# 代码重新设置文本框的背景色，不过这样比较大地增加了服务器端程序的复杂度。

退一步说，即使服务器端使用 C# 代码设置各种状态数据，那么状态数据保存在哪里呢？上面介绍了两种方式，一种是保存在客户端浏览器的 Cookie 数据包中，但 Cookie 数据包容量有限，很多浏览器的 Cookie 数据包限制为 1024 字节，非常小；另一种就是保存在服务器的 Session 中，但不管什么数据都保存在 Session 中是不妥的，这是要占用服务器内存的。Web 开发人员应当谨记，服务器资源不是个人的，而是大家共享的，不能写出霸气外露的程序。

为了解决控件状态数据无法维持的问题，ASP.NET 提供了视图状态（ViewState）技术。其原理如图 9-17 所示。

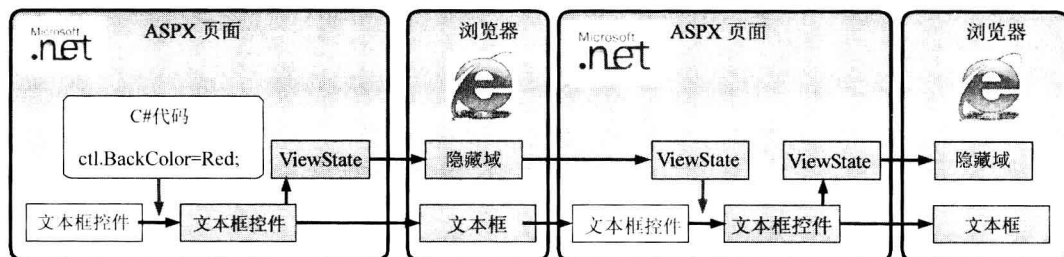


图 9-17 系统运行过程

使用 ViewState 技术，系统运行步骤如下。

第一步，在服务器端，有一段 C# 代码用来修改文本框控件的背景色为红色。控件自动地将背景色数值保存到一个 ViewState 数据包中。系统输出 HTML 代码时，ViewState 中的数据也会自动输出到 HTML 文档中的隐藏域（HTML 代码为“`<input type="hidden" />`”）中，然后发往浏览器。

第二步，浏览器接收 HTML 代码并显示出一个红色的文本框，但隐藏域是不可见的，仅仅是一个数据容器。

第三步，浏览器发起第二次 HTTP 会话，将表单数据发往服务器端，包括隐藏域中的数据。

隐藏域是标准的 HTML 元素，各种浏览器都支持。

第四步，服务器根据 ASPX 页面代码生成文本框 Web 控件对象，该文本框控件还是默认的白色背景色；系统还接收浏览器发送来的表单数据，对其中隐藏域的数据还原出 ViewState 数据包。文本框控件自动从 ViewState 数据包中检索并读取背景色信息，设置控件的背景色为红色。

第五步，服务器输出 HTML 代码，其中包括具有红色背景的文本框的 HTML 代码和包含 ViewState 数据的隐藏域 HTML 代码。

第六步，浏览器接收 HTML 代码并展现出一个红色背景的文本框，还要维持 ViewState 数据的隐藏域。

从上面的步骤可以看出，在 ASP.NET 中使用一个隐藏域维持 ViewState 数据包，实现了在多次无状态的 HTTP 会话中保持会话状态数据的目的。这个过程是自动执行的，无须开发人员照顾，因此在一定程度上降低了 Web 开发的难度。

不过使用 ViewState 技术会在 HTML 文档中插入一个隐藏域，当 ViewState 数据包比较大时，比如一个 DataTable、图像等，系统生成的隐藏域的 HTML 代码量也很大，使整个 HTML 页面代码量大，可能有几百 KB 甚至几 MB，导致系统运行缓慢，因此不能滥用 ViewState 技术。合理使用这种技术是开发人员必须考虑的问题。

每个 Web 控件各自负责自己的 HTML 代码的生成、支持服务器端事件、使用 ViewState 数据包保持控件的状态信息，这三种技术形成了 ASP.NET Web 服务器控件技术。使用 Web 服务器控件技术具有以下好处。

(1) 降低页面代码量，促成 ASP.NET 程序模块化。使用 Web 服务器控件，使得开发人员无须在页面代码中堆积代码来生成所需的 HTML 代码；而使用 Web 控件来分摊工作量，程序能够模块化，可降低程序复杂度。

(2) 代码和组件重用。ASP.NET 已经包含了大量预定义功能强大的、使用方便的 Web 控件，而且业界还存在大量的第三方 Web 控件，借助这些 Web 控件，开发人员可以方便地开发出功能强大的 Web 应用程序，轻松实现丰富的用户界面，提高开发速度。

9.1.6 ASP.NET 客户端开发架构

相对于 ASP.NET 服务器端架构，ASP.NET 客户端开发架构就很单薄了，功能有限。图 9-18 就是 ASP.NET 客户端开发架构。

在 ASP.NET 客户端开发中存在两种技术，一种是 DHTML 技术，另一种是 .NET 智能客户端技术。

DHTML (Dynamic HTML) 就是动态 HTML 技术，它是脚本语言利用浏览器提供的接口来操作 HTML 文档的内容，使之具有动态效果的技术。

在 DHTML 技术中，脚本语言一般是 JavaScript，也有采用 VBScript 的，它运行在一种受限环境中，功能受到很大限制，只能访问浏览器提供的接口，不能访问操作系统提供的接口，不能

读写文件、注册表，不能直接操作网络。由于 JavaScript 运行时限制太多了，因此微软提供了 ActiveX 控件技术，它能提升 JavaScript 的操作权限，这样做虽然能增强 JavaScript 的功能，但也带来了安全隐患。目前因特网上大量的安全问题都是 ActiveX 技术导致的。

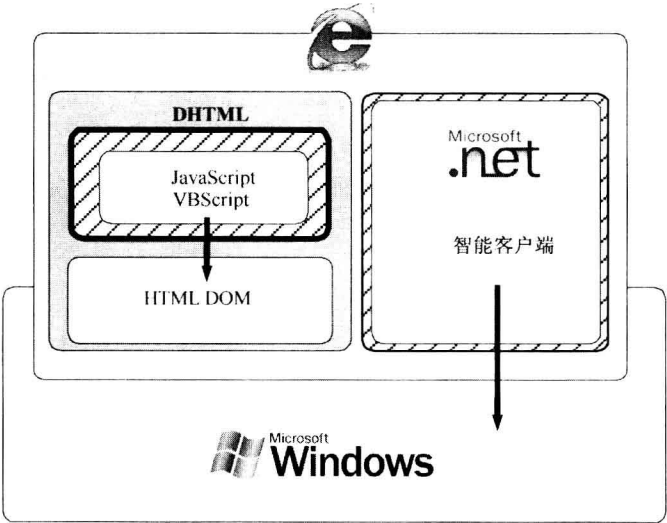


图 9-18 ASP.NET 客户端开发架构

在 DHTML 中，浏览器提供的接口就是 HTML DOM，浏览器能解析 HTML 文档内容，并根据其树状的结构创建一个树状的对象结构。JavaScript 能访问这个树状结构，获得 HTML 文档中的内容，并能修改这个树状结构，浏览器会监控对这个对象树的修改来实时地更新浏览器中显示的内容。这样 JavaScript 就能实现 HTML 页面内容的动态效果，比如增加或删除某些内容，修改文本颜色等。

.NET 智能客户端技术是 ActiveX 控件的升级版，就是让基于 .NET 技术开发的 WinForm 控件嵌入在 HTML 页面中运行。与 ActiveX 控件相比，智能客户端技术有两点不同。

- (1) 智能客户端技术必须在客户端安装微软 .NET 框架。这在应用中带来不小的麻烦，严重限制了智能客户端技术的普及应用。
- (2) 智能客户端技术受到微软 .NET 平台安全限制，以比较低的权限运行，例如不能读取文件系统中重要部分的内容等。而 ActiveX 运行是不受控制的，安全隐患很多。

在实践中，ASP.NET 客户端开发中一般用 DHTML 技术，也就是用 JavaScript 编程。

9.2 建立 C# ASP.NET 应用程序项目

使用 C# 开发 ASP.NET 应用程序不太复杂。在 VS.NET 中单击菜单项目“文件→新建→项目”，显示出如图 9-19 所示的“新建项目”对话框。

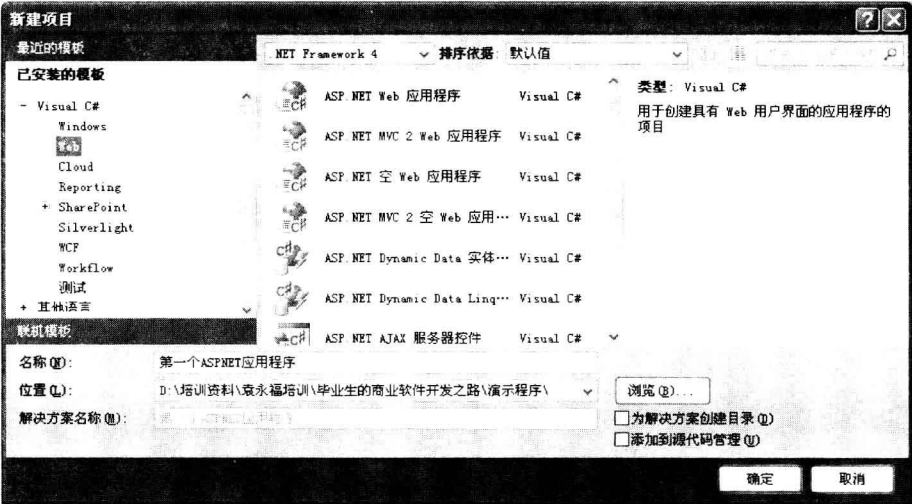


图 9-19 “新建项目”对话框

在该对话框中，在模板列表中选中“Visual C#→Web”节点，在右边的列表中选中“ASP.NET Web 应用程序”项目类型，然后输入项目名称和保存目录，单击“确定”按钮即可新建一个 C#的 ASP.NET Web 应用程序。该应用程序的 VS.NET 的用户界面如图 9-20 所示。



图 9-20 用户界面

在该用户界面中，文档编辑区显示的是默认页面 Default.aspx 的设计器界面。

对于刚刚新建的 ASP.NET Web 应用程序，其解决方案资源结构如图 9-21 所示。该结构中包含的主要节点有：

(1) Account 目录：这是进行用户管理的样板页面。

(2) App_Data 目录：这是放置数据库的专用目录。这个目录受到保护，客户端浏览器是无权限访问这个目录的，因此建议所有的数据库文件，比如 MDB 文件等，都放置在这个目录下，以保护应用程序的数据；而数据库文件放置在其他目录下都有可能被用户下载，不安全。

(3) Scripts 目录：这是放置客户端用的脚本文件目录，主要是扩展名为 js 的文件。

(4) Styles 目录：这是方式 CSS 样式表文件的目录，主要是扩展名为 css 的文件。

(5) About.aspx 页面：本系统说明信息的样板页面。

(6) Default.aspx 页面：系统主页面，应用系统运行时首先显示这个页面。

(7) Global.asax 文件：系统全局对象文件。

(8) Site.Master 文件：站点页面母模板页面，站点中所有 ASPX 页面都是用它作为母模板的。

(9) Web.config 文件：系统配置文件，可以保存数据库连接字符串之类的信息，这个文件也受到保护，不会被用户下载。

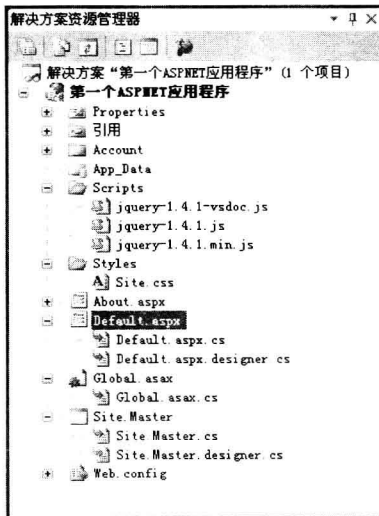


图 9-21 解决方案资源管理器

在 ASP.NET 应用程序中，最重要的文件是扩展名为 ASPX 的文件，这种文件定义了应用程序中每一个 Web 页面，是程序的功能单元，因此开发 ASP.NET 应用程序时，其中大部分工作就是设计 ASPX 页面并编写后台 C# 代码。

每个 ASPX 文件下面都跟着 .cs 文件和 .designer.cs 文件，这就是页面配套的 C# 源代码文件。例如，对于 Default.aspx 页面配备有 Default.aspx.cs 和 Default.aspx.designer.cs 两个 C# 源代码文件。其中 Default.aspx.cs 是供开发者编写功能性代码的；而 Default.aspx.designer.cs 是系统自动生成的代码文件，主要用于包含声明页面上控件变量的 C# 代码。

9.3 ASP.NET 页面设计器

在 ASP.NET 应用程序开发中，最重要的工作就是 ASP.NET 页面设计，它是应用程序的主要用户界面。VS.NET 提供 Web 页面设计器，其用户界面如图 9-22 所示。



图 9-22 ASP.NET 页面设计器

9.3.1 ASP.NET 页面工具箱

在 ASP.NET 页面设计器中，左边是 ASP.NET 页面工具箱，列出了所有可用的 ASP.NET Web 控件。设计者可以将控件拖曳到右边的页面中，或者双击控件在页面的当前插入点处添加新的 Web 控件。

微软为了提高 Web 开发效率，提供了 ASP.NET Web 控件的功能，这些 Web 控件可以拖曳，修改大小，可以在属性编辑器中编辑属性，可以有方法、事件；这些和 Windows 控件有很多类

似的地方。微软也尽量增加 Web 开发和 Windows 窗体开发的共同点，使得开发者能在两者之间较为方便地切换。

经过实践，发现使用 Web 控件确实能降低 Web 开发的难度，提高开发效率，因此现在有不少其他的 Web 开发技术也开始吸收 Web 控件的概念。

9.3.2 Web 页面内容编辑器

ASP.NET 页面设计器的右边是 Web 页面内容编辑器，在图 9-22 中显示的 Web 页面是应用了母板页面的样式。

这个设计器有点像 Dreamwave 之类的网页内容设计软件，用户可以直接在这个编辑器中输入文字、插入图片、放置表格，或者将 Web 控件从工具箱上拖曳到页面中。

在页面设计器的下方有一个工具条，有“设计”、“拆分”、“源”三个按钮，还有页面当前插入点所在的 HTML 层次。

Web 页面设计器有三种显示模式：设计显示模式、拆分显示模式和源代码显示模式。图 9-22 中是设计显示模式，单击设计器下面工具条上的“拆分”按钮即可切换到拆分显示模式，此时用户界面如图 9-23 所示。

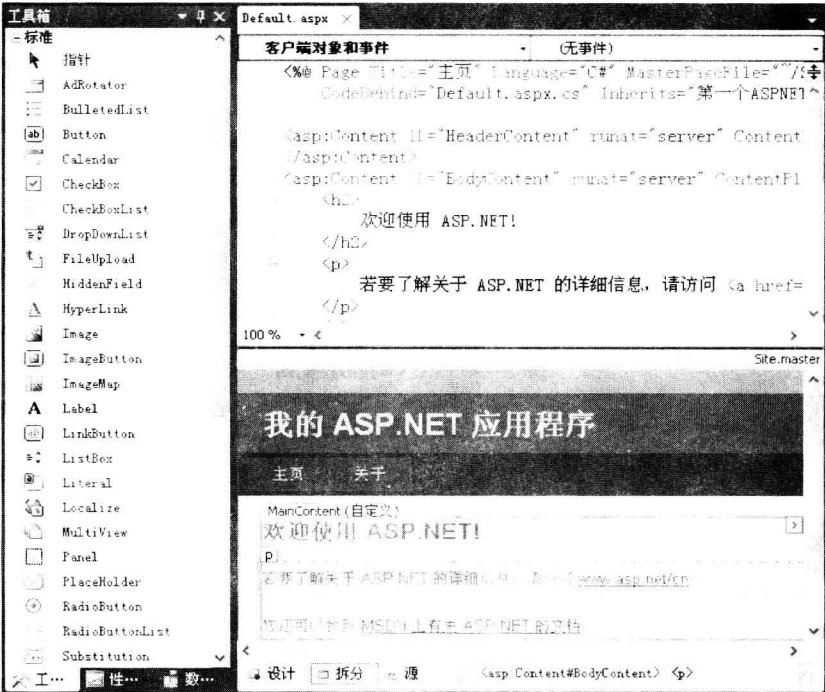


图 9-23 拆分显示模式

在拆分显示模式下，编辑器上半部分是页面的 HTML 源代码编辑区，下半部分是页面编辑

区。设计者可以在源代码文本编辑区或页面编辑区中编辑页面内容，双方的内容是同步更新的，而且内容选择状态也是同步的。

例如，可在图 9-23 下方的页面编辑区中选择一段文本，然后在上方的源代码编辑区中同步选择相关的 HTML 源代码。

拆分显示模式很适合进行页面的精细调整。

单击设计器下方工具条上的“源”按钮，设计器就切换到源代码显示模式，其用户界面如图 9-24 所示。

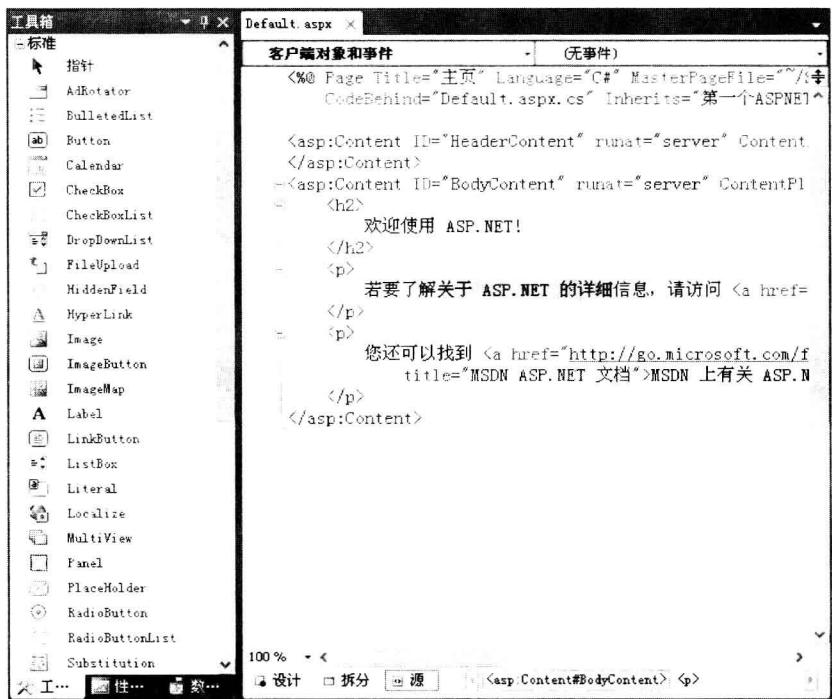


图 9-24 源代码显示模式

在源代码显示模式下，设计者可以输入和编辑大段 HTML 源代码。

9.4 ASP.NET 控件工具箱

设计器右边是 ASP.NET 控件工具箱，上面列出了所有可用的控件。开发者可以双击工具箱中的某个控件按钮来向 ASP.NET 页面插入点处插入一个控件，或者用鼠标拖曳一个控件项目到 ASP.NET 页面中新增一个控件。

例如双击工具箱中的按钮项目 “ Button”，则在 ASP.NET 页面中新增一个按钮控件，若页面设计器处于源代码模式下，则在 HTML 源代码中插入代码块 “<asp:Button

ID="Button1" runat="server" Text="Button" />"; 若页面设计器处于设计模式, 则在页面中插入如图 9-25 所示的按钮控件。在设计模式中, 用户单击按钮控件, 则在左边的对象属性编辑器中可以编辑控件的一些属性和事件。

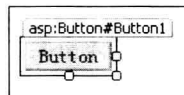


图 9-25 按钮控件

9.5 设计用户界面

下面为这个 ASP.NET 应用程序设计用户界面。在解决资源管理器中双击节点 Default.aspx 打开该页面的页面设计器, 切换到设计模式。

从工具箱中拖曳一个 TextBox 控件到页面中, 在属性编辑器中设置控件 ID 为 txtNumber1。

拖曳一个 DropDownList 控件到页面中, 在属性编辑器中设置控件 ID 为 cboOperator, 选中它的 Items 属性, 打开该属性值的编辑对话框, 如图 9-26 所示添加加、减、乘、除 4 个字符的选项。

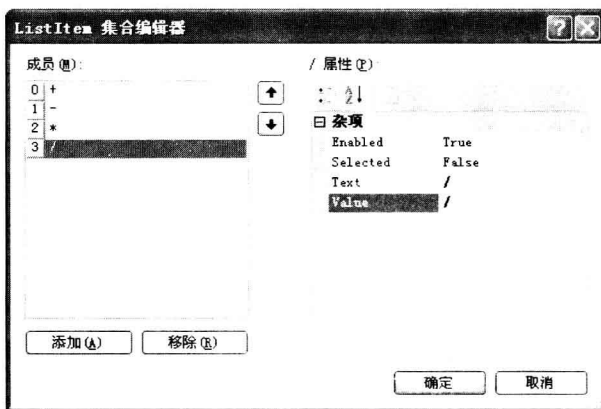


图 9-26 编辑对话框

继续添加一个 TextBox 控件, 设置 ID 为 txtNumber2; 添加一个 Button 控件, 设置 ID 值为 btnCalculate, Text 属性值为 "="; 添加一个 TextBox 控件, 设置 ID 值为 txtResult, Readonly 属性值为 true。这样就完成了本软件的用户界面的设计。

9.6 编写后台代码

本程序的功能是用户单击 "=" 按钮即可进行加、减、乘、除的计算, 因此后台代码的功能就是响应按钮的点击事件进行数值运算。

在页面的设计界面中双击 "=" 按钮, 则系统切换到该页面的 C# 代码编辑器, 并自动生成一个名为 "btnCalculate_Click" 的按钮的点击事件处理方法体, 开发人员可以在这个方法体中插入功能性代码。

首先观察 C# 代码的整体结构，一个 ASPX 文件配有两个 C# 代码文件，一个名为“页面名称.aspx.cs”，用于放置功能性代码；另一个名为“页面名称.aspx.designer.cs”，用于定义页面中控件的变量。

对于本页面，文件名为“Default.aspx”，功能性 C# 代码文件名为“Default.aspx.cs”，其内容如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace 第一个 ASPNET 应用程序
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

还有一个名为“Default.aspx.designer.cs”，其内容如下：

```
//-----
// <自动生成>
//     此代码由工具生成
//
//     对此文件的更改可能会导致不正确的行为，并且如果
//     重新生成代码，这些更改将会丢失
// </自动生成>
//-----

namespace 第一个 ASPNET 应用程序 {

    public partial class _Default {

        /// <summary>
        /// txtNumber1 控件
        /// </summary>
        /// <remarks>
        /// 自动生成的字段
        /// 若要进行修改，请将字段声明从设计器文件移到代码隐藏文件
        /// </remarks>
        protected global::System.Web.UI.WebControls.TextBox txtNumber1;

        /// <summary>
        /// cboOperator 控件
        /// </summary>
        /// <remarks>
        /// 自动生成的字段
        /// 若要进行修改，请将字段声明从设计器文件移到代码隐藏文件
    }
}
```

```

    /// </remarks>
    protected global::System.Web.UI.WebControls.DropDownList cboOperator;

    /// <summary>
    /// txtNumber2 控件
    /// </summary>
    /// <remarks>
    /// 自动生成的字段
    /// 若要进行修改, 请将字段声明从设计器文件移到代码隐藏文件
    /// </remarks>
    protected global::System.Web.UI.WebControls.TextBox txtNumber2;

    /// <summary>
    /// btnCalculate 控件
    /// </summary>
    /// <remarks>
    /// 自动生成的字段
    /// 若要进行修改, 请将字段声明从设计器文件移到代码隐藏文件
    /// </remarks>
    protected global::System.Web.UI.WebControls.Button btnCalculate;

    /// <summary>
    /// txtResult 控件
    /// </summary>
    /// <remarks>
    /// 自动生成的字段
    /// 若要进行修改, 请将字段声明从设计器文件移到代码隐藏文件
    /// </remarks>
    protected global::System.Web.UI.WebControls.TextBox txtResult;
}
}

```

在这个 C#代码中定义了页面中所有的 Web 控件变量。当用户在 ASPX 页面中新增、删除 Web 服务器控件或修改控件的名称时, 这个代码也会被自动更新来同步设计。

可以看出这个对象类型继承自 `System.Web.UI.Page`, 类型全名为“第一个 ASPNET 应用程序._Default”。

打开 ASPX 页面的 HTML 代码, 可以看到第一行代码如下:

```

<%@Page Title="主页" Language="C#" MasterPageFile="~/Site.master"
AutoEvent Wireup="true" CodeBehind="Default.aspx.cs"
Inherits="第一个 ASPNET 应用程序._Default" %>

```

这段代码中“`CodeBehind="Default.aspx.cs"`”说明本 ASPX 页面绑定的功能性 C#代码文件为“`Default.aspx.cs`”, 能帮助 VS.NET 找到对应的 C#代码文件, 对程序运行本身没有作用;“`Inherits="第一个 ASPNET 应用程序._Default"`”说明本页面绑定的对象类型为“第一个 ASPNET 应用程序._Default”, 这是 ASP.NET 页面能运行的基础设置。

在“`btnCalculate_Click`”中编写功能性代码, 代码如下:

```

protected void btnCalculate_Click(object sender, EventArgs e)
{
    try
    {
        double number1 = Convert.ToDouble(txtNumber1.Text);
    }
}

```

```
double number2 = Convert.ToDouble(txtNumber2.Text);
double result = 0;
switch (cboOperator.Text)
{
    case "+":
        result = number1 + number2;
        break;
    case "-":
        result = number1 - number2;
        break;
    case "*":
        result = number1 * number2;
        break;
    case "/":
        result = number1 / number2;
        break;
}
txtResult.Text = result.ToString();
}
catch (Exception ext)
{
    string script = "<script language='javascript'>alert('"
        + HttpUtility.JavaScriptStringEncode(ext.Message)
        + "');</script>";
    this.ClientScript.RegisterStartupScript(this.GetType(), "Error",
script);
}
}
```

这样就完成了本程序的开发，可以编译执行了，其程序运行时的效果如图 9-27 所示。

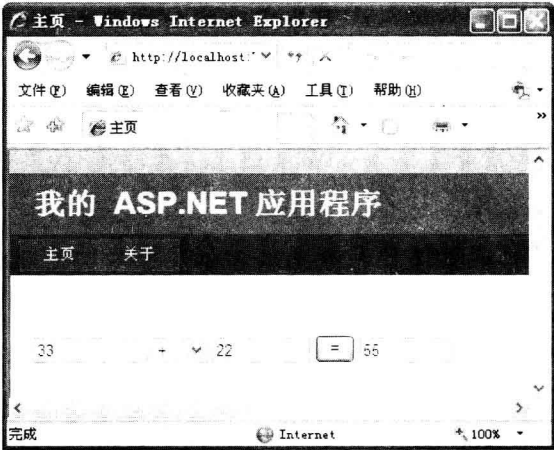


图 9-27 程序运行效果

9.6.1 输出 JavaScript

笔者以前曾开发过 WinForm 版本的计算器程序，其命令按钮事件处理代码如下：

```
private void btnCalculate_Click(object sender, EventArgs e)
{
```

```

try
{
    double number1 = Convert.ToDouble(txtNumber1.Text);
    double number2 = Convert.ToDouble(txtNumber2.Text);
    double result = 0;
    switch (cboOperator.Text)
    {
        case "+":
            result = number1 + number2;
            break;
        case "-":
            result = number1 - number2;
            break;
        case "*":
            result = number1 * number2;
            break;
        case "/":
            result = number1 / number2;
            break;
    }
    txtResult.Text = result.ToString();
}
catch (Exception ext)
{
    MessageBox.Show(ext.Message);
}
}

```

经对比可以看出两者非常接近，只是显示错误消息的代码有所不同。对于 WinForm 程序，显示错误的代码如下：

```

MessageBox.Show(ext.Message);

```

而对于 ASP.NET 程序，显示错误的代码如下：

```

string script = "<script language='javascript'>alert('"
    + HttpUtility.JavaScriptStringEncode(ext.Message)
    + "');</script>";
this.ClientScript.RegisterStartupScript(this.GetType(), "Error",
script);

```

在 ASP.NET 中不能直接显示用户界面，不能使用 MessageBox 对话框来显示消息。因为在实际运行中，ASP.NET 程序是以服务方式运行的，此时若显示用户界面也是孤芳自赏，任何人都看不到，程序会一直等待用户关闭对话框继续显示，而实际上没有任何用户能看到或者关闭对话框，此时程序会永远地暂停了。对用户来说，B/S 系统相当于死机了，没有任何反应，这是一种比较严重的程序功能错误，必须避免。因此 ASP.NET 程序中绝对要避免显示对话框等用户界面。

ASP.NET 程序唯一展现用户界面的方式就是生成 HTML 代码让浏览器显示出来，而在 HTML 代码中，可以加上 JavaScript 代码让客户端浏览器来显示消息框。

开发人员可以在设计 ASP.NET 页面时直接在 ASPX 文件中的 HTML 代码中加上 JavaScript 代码，也可以使用 C# 程序动态地生成 JavaScript 代码文本插入到 HTML 文档中。此处就是动态生成 JavaScript 代码文本。

以下 C# 代码能生成客户端执行的 JavaScript 代码文本。

```
string script = "<script language='javascript'>alert('"  
    + HttpUtility.JavaScriptStringEncode(ext.Message)  
    + "')</script>";
```

例如，可以生成 “<script language='javascript'>alert('输入字符串的格式不正确。')</script>”，这里调用的函数 “HttpUtility.JavaScriptStringEncode” 能将任意字符串转换为符合 JavaScript 语法的字符串内容。

所有 ASP.NET 页面都派生自类型 System.Web.UI.Page，Page 类型有个名为 ClientScript 的属性能获得一个对象，这个对象有两个用得最多的成员方法，其定义如下：

方法 1：

```
public void RegisterStartupScript(Type type, string key, string script);
```

方法 2：

```
public void RegisterClientScriptBlock(Type type, string key, string  
script);
```

这两个方法都能将 JavaScript 代码输出到 HTML 文档中。这两个方法的参数如下：

Type type: 要输出的 JavaScript 脚本文本的类型。一般为页面或 Web 控件的类型。

string key: 输出脚本的名称。使用这个名称，即使代码重复调用几次也不会导致多次输出相同的代码。

string script: 要输出的脚本代码文本，应该包含 “<script language="javascript"> </script>”。从理论上来说是可以包含任何 HTML 代码的，而不限于 JavaScript。

RegisterStartupScript 和 RegisterClientScriptBlock 方法都能输出 JavaScript 代码，但输出的内容是不同的，其效果也不同，其原理如图 9-28 所示。

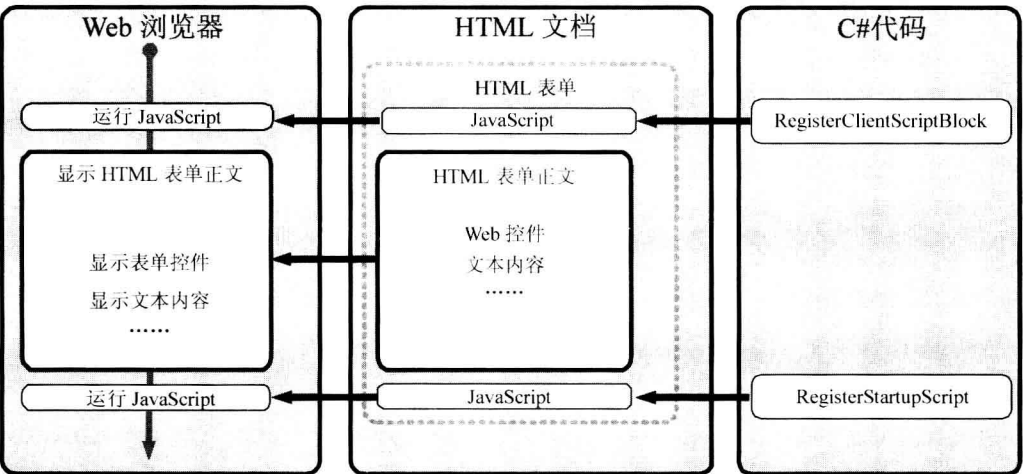


图 9-28 原理图

对于 RegisterClientScriptBlock 方法，它会在 HTML 文档的开头处所有 HTML 文档正文内容

之前输出 JavaScript 代码。而客户端浏览器是将执行 JavaScript 代码和加载显示 HTML 文档 enrico 串行执行的，客户端浏览器在执行 JavaScript 代码时不会加载后续的 HTML 文档内容。

此时客户端浏览器首先运行 RegisterStartupScript 方法插入的 JavaScript 代码，然后再加载显示 HTML 表单内容。此时若 JavaScript 代码中弹出一个对话框等待用户输入，则浏览器暂停操作，此时用户看不到任何文档内容。

对于 RegisterStartupScript 方法，它会在 HTML 文档的末尾处输出 JavaScript 代码，准确地说，是在页面表单元素结束符号 “</form>” 前输出代码。客户端浏览器运行到这段代码时已经加载显示了 HTML 文档中的可视部分。

对于本 ASP.NET 程序，当使用 RegisterStartupScript 方法输出 JavaScript 时，若程序发生错误，比如在 txtNumber1 中输入了非数字字符 “aaa”，则程序运行界面如图 9-29 所示。



图 9-29 程序运行界面

当使用 RegisterClientScriptBlock 方法输出 JavaScript 时，若程序发生错误，比如在 txtNumber1 中输入了非数字字符 “aaa”，则程序运行界面如图 9-30 所示。

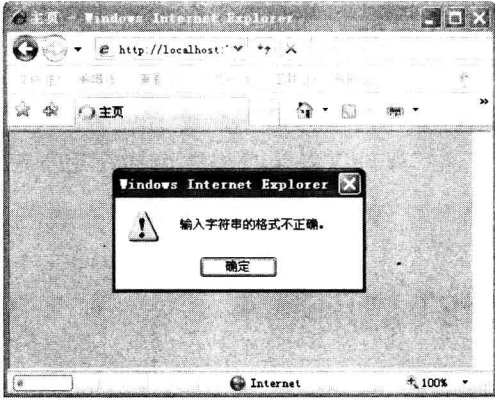


图 9-30 程序运行界面

只有当用户单击“确定”按钮关闭消息框时，浏览器才会接着加载 HTML 文档的后续内容，加载完毕后的用户界面如图 9-31 所示。

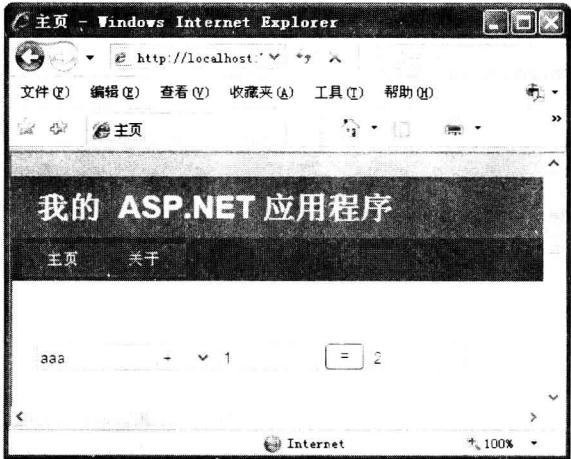


图 9-31 用户界面

很显然，应该采用“RegisterStartupScript”方法，因为使用它时的用户体验比“RegisterClientScriptBlock”好。

9.7 测试和运行 ASP.NET 应用程序

在 VS.NET 中测试和运行 ASP.NET 应用程序是很简单的，可采用两种方式运行页面。

第一种是以调试模式启动。首先设置启动页面，具体做法就是在解决方案资源管理器中选中 Default.aspx 节点，然后用鼠标右击弹出快捷菜单，单击“设为起始页”即可。按下快捷键 F5 就能以调试模式启动应用程序，打开的第一个页面就是系统的起始页。

当以调试模式运行 ASP.NET 程序时，开发人员可以在 C# 代码中设置断点、可以单步调试、可以查看修改变量名、可以修改 C# 代码后继续执行，用起来很方便。

第二种就是在解决方案资源管理器中选中 Default.aspx 节点，然后用鼠标右击弹出快捷菜单，单击“在浏览器中查看”就可以以运行模式启动应用程序，打开的第一个页面是“Default.aspx”。

9.8 部署 ASP.NET 应用程序

ASP.NET 应用程序开发完毕后，需要脱离 VS.NET 而独立运行，其过程如下。

9.8.1 准备运行环境

部署 ASP.NET 应用程序首先要准备运行环境，要确定部署 ASP.NET 应用程序的服务器系统已安装了 IIS 和微软.NET 框架。打开 Windows 服务管理器，以确保“World Wide Web Publishing”服务处于运行状态，如图 9-32 所示。

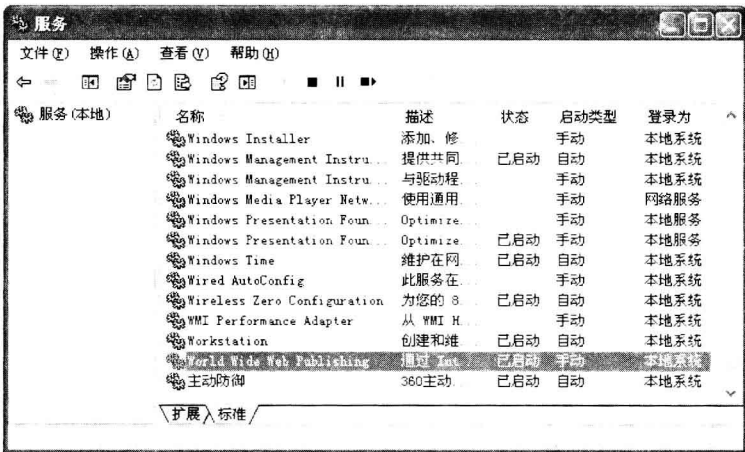


图 9-32 Windows 服务管理器

如果没有发现“World Wide Web Publishing”服务，说明系统中没有安装 IIS，需要用 Windows 安装光盘安装 IIS 组件。

9.8.2 准备应用程序目录

打开 Windows 资源管理器，将 ASP.NET 应用程序文件复制到某个文件目录下。此时就可以删除所有扩展名为“.cs”和“.csproj”的程序源代码文件了。

9.8.3 创建虚拟目录

部署 ASP.NET 应用程序的第二步就是创建虚拟目录。

所谓虚拟目录就是在 IIS 上建立的一个虚拟的文件目录，这个虚拟文件目录和计算机文件系统中的某个实际存在的目录具有映射关系，IIS 能将客户端访问虚拟目录的动作映射到服务器文件系统中的指定目录。

打开 Windows 控制面板中的管理工具下面的 Internet 信息服务软件，其用户界面如图 9-33 所示。



图 9-33 Internet 信息服务

在该用户界面中，用鼠标右击“默认网站”节点，单击快捷菜单“新建→虚拟目录”，弹出如图 9-34 所示的“虚拟目录创建向导”对话框。

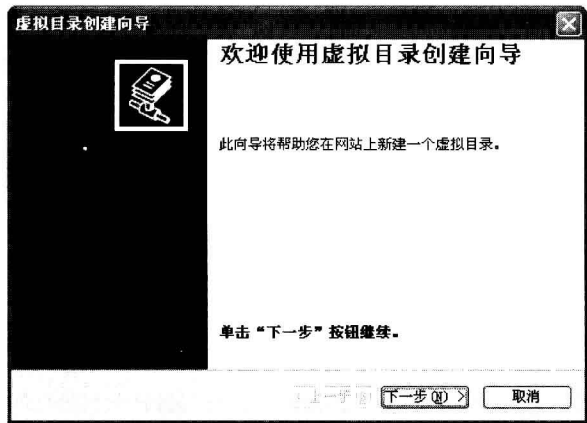


图 9-34 “虚拟目录创建向导”对话框

在该对话框中单击“下一步”按钮，在弹出的对话框中输入虚拟目录的别名，如图 9-35 所示。

所谓虚拟目录的别名就是这个 B/S 应用程序的发布名称，在此输入“FirstASPNET”。以后可以通过“http://服务器名称/FirstASPNET”的 URL 地址来访问应用程序。

单击“下一步”按钮，在如图 9-36 所示的对话框中输入应用程序的本地文件目录，在此输入“第一个 ASPNET 应用程序”所在的完整目录地址。

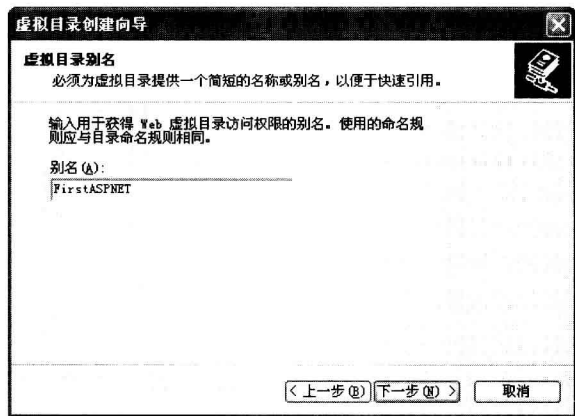


图 9-35 “虚拟目录别名”对话框

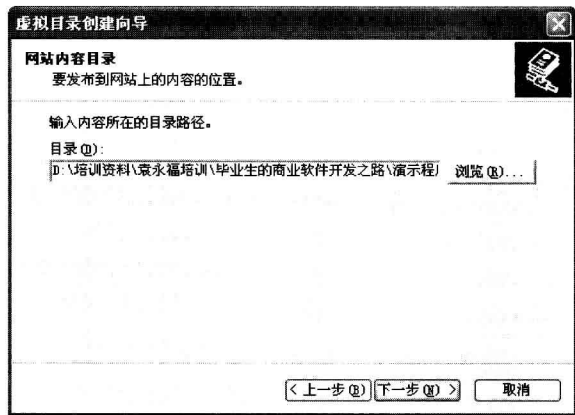


图 9-36 “网站内容目录”对话框

单击“下一步”按钮，在如图 9-37 所示的对话框中设置应用程序的一些权限。

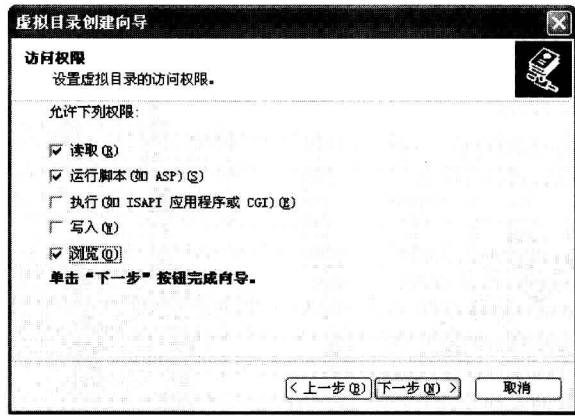


图 9-37 “访问权限”对话框

单击“下一步”按钮，即可完成发布 ASP.NET 应用程序的初步设置，如图 9-38 所示

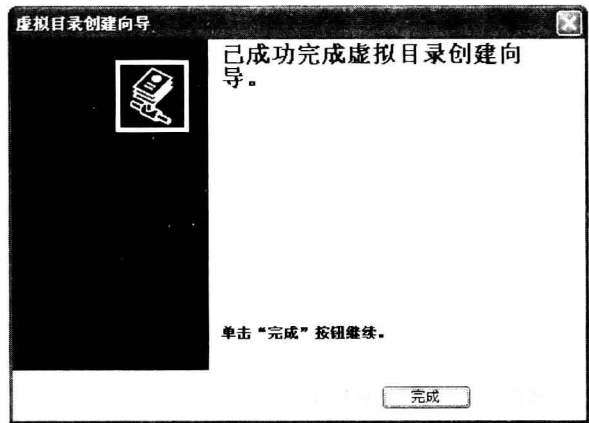


图 9-38 完成虚拟目录的创建

完成所有的操作后，如图 9-39 所示，“默认网站”节点下就新增了一个“FirstASPNET”节点。

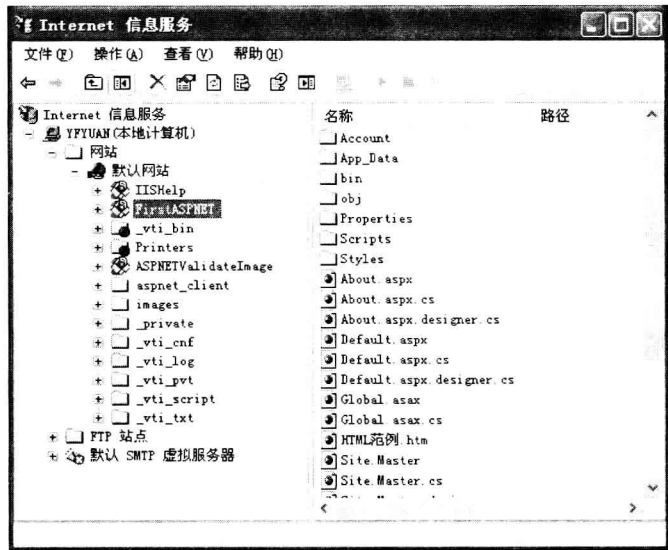


图 9-39 Internet 信息服务

其实还有一种很快捷的方法可以设置虚拟目录。打开 Windows 资源管理器，将 ASP.NET 应用程序文件复制到某个文件目录下。此时可以删除所有扩展名为“.cs”和“.csproj”的程序源代码文件。用鼠标右击这个文件目录，弹出快捷菜单，单击“属性”菜单项目，打开如图 9-40 所示的文件目录属性对话框。

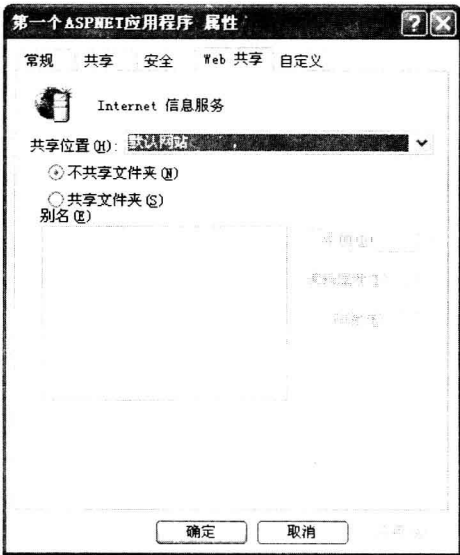


图 9-40 文件目录属性对话框

在该对话框中单击“共享文件夹”单选按钮，弹出如图 9-41 所示的对话框。

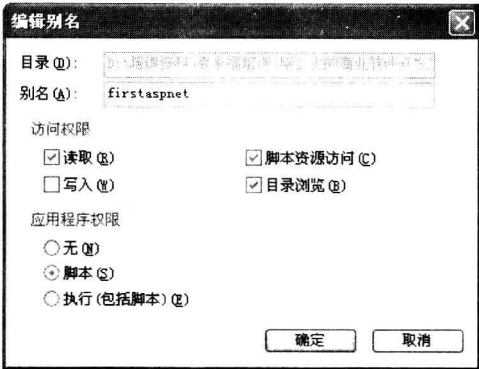


图 9-41 “编辑别名”对话框

在该对话框中单击“确定”按钮关闭对话框，则本目录已经被设置为 Web 目录，可以使用“http://localhost/firstaspnet”访问了。

注意，对于 IIS，虚拟目录名是不区分大小写的，而且同一台计算机中不能有两个同名的虚拟目录。

9.8.4 配置虚拟目录

创建网站的虚拟目录后还不能立即使用系统，还需要进行一些站点的配置。

1. 站点权限

在很多情况下都需要配置站点权限，否则可能出现站点无权访问的问题。设置站点权限，如图 9-42 所示。

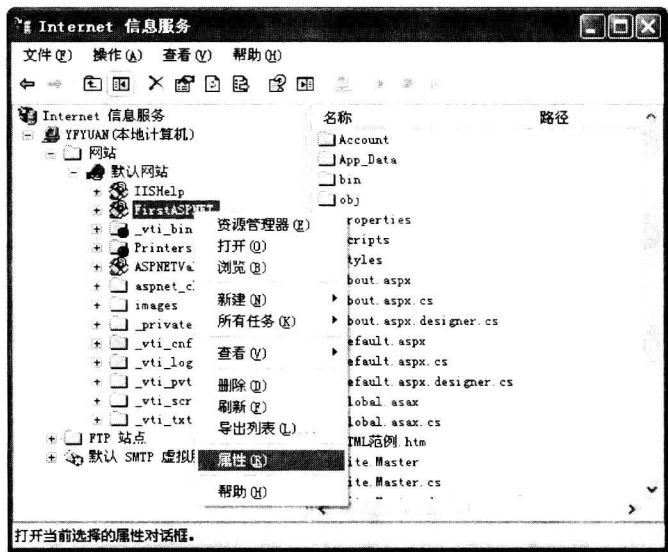


图 9-42 设置站点权限

在“Internet 信息服务”管理器中，用鼠标右击“FirstASPNET”节点，弹出快捷菜单，然后单击“属性”菜单项目，弹出如图 9-43 所示的站点属性对话框。

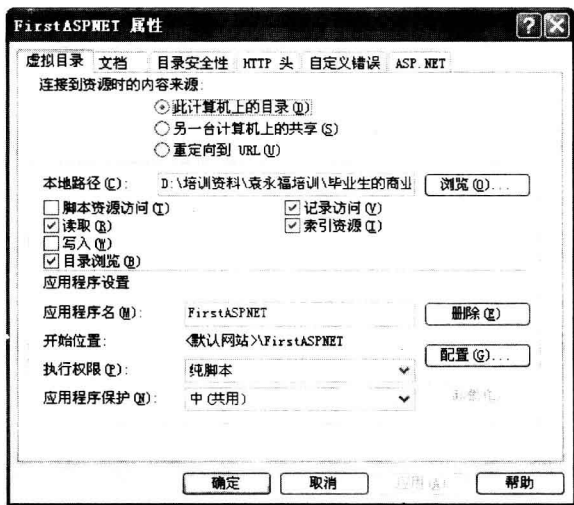


图 9-43 站点属性对话框

切换到“目录安全性”选项卡，就可以看到如图 9-44 所示的用户界面。

单击其中的“编辑”按钮会弹出如图 9-45 所示的对话框。

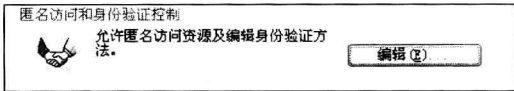


图 9-44 用户界面

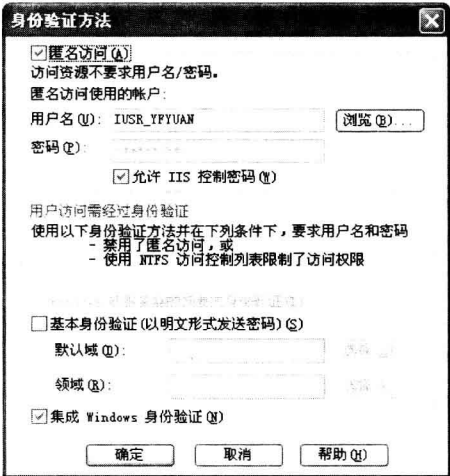


图 9-45 “身份验证方法”对话框

在该对话框中勾选“匿名访问”，输入用户名。

2. ASP.NET 版本号

当服务器系统安装了多个版本的微软.NET 框架时，需要根据 ASP.NET 应用程序使用的.NET 框架版本号来设置站点使用的.NET 框架版本号。

如图 9-46 所示，在站点属性对话框中切换到“ASP.NET”选项卡。

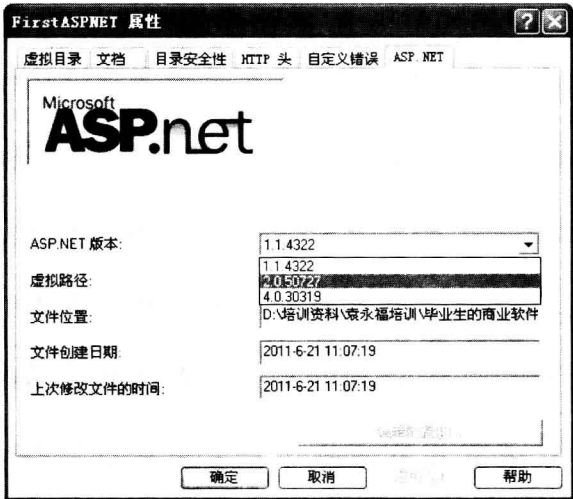


图 9-46 “ASP.NET”选项卡

在该选项卡中的“ASP.NET 版本”下拉列表中选择 ASP.NET 应用程序所用的 .NET 框架版本号。

9.9 其他部署相关技术手段

在部署 ASP.NET 应用程序中有一些其他相关技术手段和工具，常用的有以下几种。

9.9.1 AspNet_regiis.exe

这是一个微软 .NET 框架包含的命令程序，被称为 ASP.NET IIS 注册工具，用于向 Internet Information Services (IIS) 注册 ASP.NET 应用程序框架。对于 ASP.NET 2.0 或更高版本，这个工具程序的地址一般为“C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727”。它的常用语法如表 9-1 所示。

表 9-1 命令程序的常用语法

命 令 行	功 能
AspNet_regiis.exe -i	安装与 aspNet_regiis.exe 相关的 ASP.NET 版本，更新 IIS 中的对 ASP.NET 应用程序文件的映射
AspNet_regiis.exe -lv	列出 IIS 中安装的所有 ASP.NET 版本号及相关信息
AspNet_regiis.exe -u	卸载 IIS 中的与 aspNet_regiis.exe 相关的 ASP.NET 版本

若在部署 ASP.NET 应用程序后发现应用程序错误，则可以考虑调用“aspnet_regiis.exe -u”和“aspnet_regiis.exe -i”来先卸载后重新安装 ASP.NET 应用程序框架，一般能解决问题。

9.9.2 IISReset.exe

IISReset.exe 是一个命令程序，用于重新启动 IIS 服务。该程序文件一般位于“c:\windows\system32”目录下。由于 Windows 系统的 Path 系统变量已经包含了这个目录，因此可以在 Windows 命令行中直接输入该命令。

若没有加上命令行参数，则该命令能关闭并重新启动 IIS 服务，一般该程序在用户界面中输出的文本如下：

```
正在尝试停止...
Internet 服务已成功停止
正在尝试启动...
Internet 已成功重新启动
```

当 ASP.NET 应用程序或 Web 站点存在问题时，使用该工具一般能解决问题。

9.9.3 配置 ASP.NET 账号权限

ASP.NET 程序使用的 Windows 用户账号名一般为“ASPNET”，对于不同的系统可能有所不同

同。该账号的权限默认是比较低的，有可能因权限不足而导致 ASP.NET 应用程序发生拒绝访问的错误。此时需要提升 ASP.NET 用户账号的权限。

一般来说，ASP.NET 应用程序对应用程序目录及其子目录具有读写文件的权限。若发生文件操作失败，则需要检查文件或目录是否设置为只读标记。

若 ASP.NET 应用程序需要访问其他文件目录，则需要对这个文件目录开通针对 ASP.NET 账号的权限。其具体做法是在 Windows 资源管理器中，用鼠标右击指定的目录，弹出属性对话框，切换到“安全”选项卡，如图 9-47 所示。

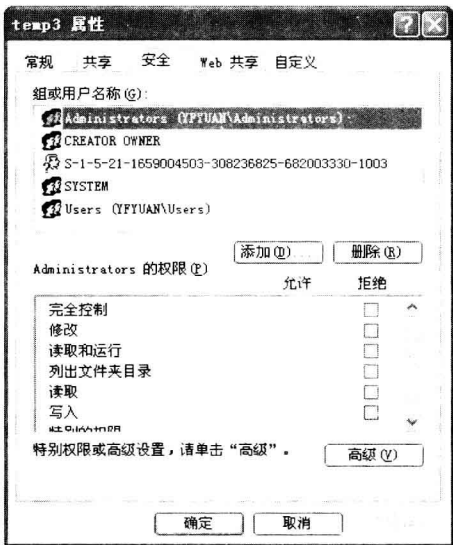


图 9-47 “安全”选项卡

在该选项卡中，单击“添加”按钮，弹出如图 9-48 所示的对话框。

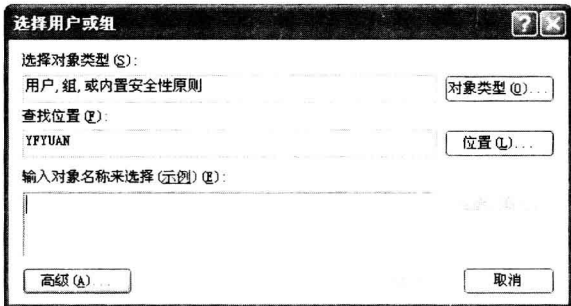


图 9-48 “选择用户或组”对话框

在该对话框中单击“高级”按钮，展开该对话框下半部分，如图 9-49 所示。

单击“立即查找”按钮，则在列表中列出了本计算机中的所有用户和用户组，选中

“ASPNET” 用户，然后单击“确定”按钮关闭对话框，回到属性对话框，如图 9-50 所示。

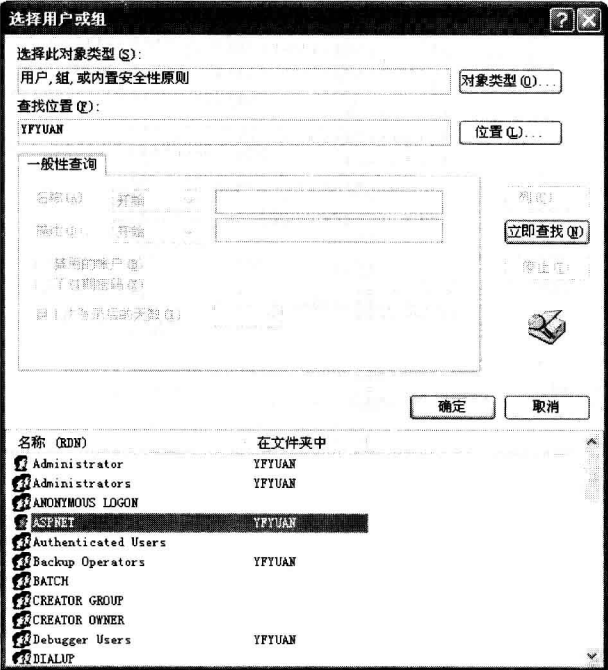


图 9-49 “选择用户或组”对话框

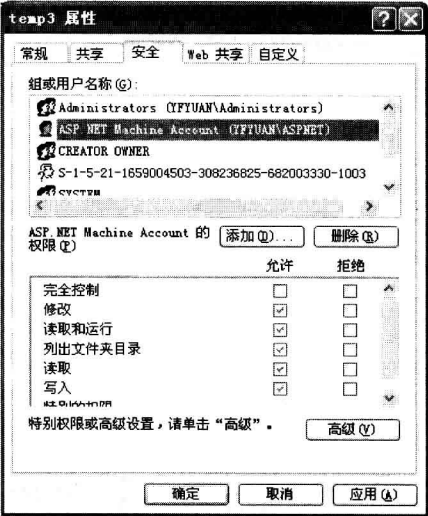


图 9-50 属性对话框

在“组或用户名称”列表中已经出现了“ASPNET”用户，在下面的权限列表中勾选“修改”、“写入”等所需权限。这样 ASP.NET 应用程序就有权访问这个目录了。

开发第一个 Web Service 程序

Web Service 是现在 Web 开发中一个很重要的应用程序类型。它以一种比较安全和可靠的方式向其他应用程序提供数据和功能。近期兴起的云计算就源自 Web Service。

10.1 Web Service 原理

图 10-1 是 Web Service 的原理图。

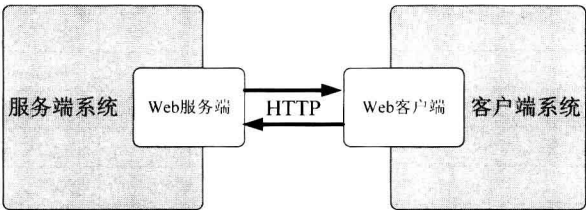


图 10-1 Web Service 的原理图

Web Service 是服务器端和客户端交流数据的方式。其运行过程如下。

- (1) 客户端应用系统决定调用 Web Service 获取功能。此时它将一些参数传递给 Web 客户端的程序组件，并等待执行结果。
- (2) Web 客户端将参数进行某种序列化，将其转换为 XML 文档。然后使用 HTTP 协议通过网络发往 Web 服务器端，等待 HTTP 响应。
- (3) Web 服务器端接收到 XML 文档，将其进行反序列化还原出参数，然后交给服务器端系统。
- (4) 服务器端系统接收参数，执行某些功能，然后将运算结果交给 Web 服务器端。
- (5) Web 服务器端将运算结果序列化成 XML 文档，将其作为 HTTP 响应结果通过 HTTP 协议返回给 Web 客户端。
- (6) Web 客户端接收 XML 文档，将其进行反序列化，还原出服务端系统运算所得的结果，然后交给客户系统。
- (7) 客户系统接收数据，继续执行。

Web Service 已经被确定为国际标准，因此在使用 Web Service 中，服务端系统和客户端系统可以是异构系统，可以分别是 Java、.NET 或 PHP 等。这样 Web Service 就成为异构系统中信息集成和交换的首选技术手段。

10.2 软件功能需求

这里提出了一个软件功能需求，某独立的系统已经有一个数据库 Customers.mdb，其中有一个数据表 Customers，该数据表存储了客户基本信息。表 10-1 列出了 Customers 数据表中的部分内容，此处没有列出所有的记录和所有的字段。

表 10-1 Customers 数据表

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
BERGS	少室山公司	方证	采购员	东园西甲 30 号	长平
BLAUS	擎天航空	雷震子	销售代表	常保阁东 80 号	莫斯科
BLONP	华夏工程	李大禹	市场经理	广发北路 10 号	幽州
BOLID	武当投资	宋青书	物主	临翠大街 80 号	巴伐利亚
BONAP	擎天南京公司	大星星	物主	花园东街 90 号	许安
BOTTM	擎天山公司	童姥	结算经理	平谷嘉石大街 38 号	许安
BSBEV	光明杂志	徐庶	销售代表	黄石路 50 号	羊城
CACTU	威航货运有限公司	鬼谷子	销售代理	经七纬二路 13 号	幽州
CENTC	二捷实业	卫青	市场经理	英雄山路 84 号	幽州
CHOPS	梁山大酒店	孙二娘	物主	白广路 314 号	莫斯科
COMMI	SinoSoft	兰陵笑笑生	销售员	七一路 37 号	莫斯科
CONSH	擎天科技	千里独行	销售代表	劳动路 23 号	虎牢

现在其他独立的应用系统需要读取客户的基本信息，但为了安全和减少系统之间的耦合性，其他应用系统不能连接数据库。因此本系统采用 Web Service 的方式向其他应用系统提供数据。

其他系统获得客户基本信息的具体技术要求是：

- (1) 能获得所有客户编号，也就是获得所有记录的 CustomerID 字段值。
- (2) 能获得指定客户编号的基本信息，也就是获得指定 CustomerID 值的记录的所有字段值。

10.3 建立 C# Web Service 应用程序项目

Web Service 底层技术还是很多的，不过.NET 框架已经掩盖了很多技术细节，使用起来很方便，而 VS.NET 对开发 Web Service 提供了很强大的支持，减少了入门难度。

在 VS.NET 中，单击主菜单项目“文件→新建→项目”，打开如图 10-2 所示的“新建项目”对话框。

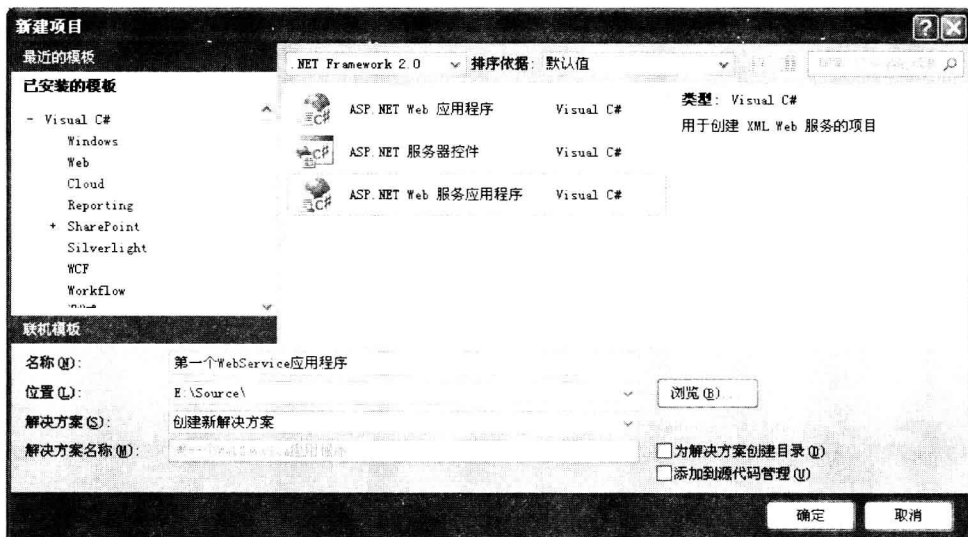


图 10-2 “新建项目”对话框

在该对话框左边的列表中选中“Web”项目，在右边列表中选中“ASP.NET Web 服务应用程序”，输入项目名称，然后单击“确定”按钮，即可创建一个 Web Service 应用程序项目。

一个刚刚创建的 Web Service 应用程序项目，其解决方案资源管理器中列出的项目如图 10-3 所示，而且 C#代码编辑器已经默认打开了文件“Service1.asmx.cs”。

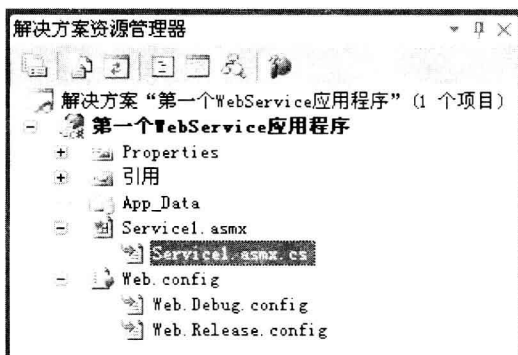


图 10-3 解决方案资源管理器

在这里以 asmx 为扩展名的就是 Web Service 页面，这种页面没有任何用户界面，只能编写 C#代码对外提供方法。

将数据库文件 Customers.mdb 复制到工程所在目录下的 App_Data 子目录中，并包含在本 C#项目中。

单击主菜单项目“项目→添加新项”，弹出如图 10-4 所示的“添加新项”对话框。



图 10-4 “添加新项”对话框

在该对话框中左边的列表中选择“Web”项目，在右边的列表中选择“Web 服务”，输入名称“CustomerService.asmx”，单击“添加”按钮，即可向项目添加一个新的 Web Service 页面，并打开该页面的 C# 代码文件。

可以看到刚刚新建的 CustomerService.asmx.cs 文件中定义了一个类型，声明该类型的代码如下：

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
public class CustomerService : System.Web.Services.WebService
```

可以看出，Web Service 功能对象都是派生自类型“System.Web.Services.WebService”的，并且前面附加了 WebService 和 WebServiceBinding 特性。

10.4 编写 Web 方法

CustomerService 是一个容器，能放置若干个 Web 方法，Web 方法就是具体实现 Web Service 功能的代码体。

在该类型中，首先编写以下代码，定义了一个连接数据库的通用成员方法。

```
/// <summary>
/// 创建一个数据库连接对象
/// </summary>
/// <returns>创建的数据库连接对象</returns>
private IDbConnection CreateConnection()
```



```

{
    return new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source="
        + this.Server.MapPath("App_Data\\Customers.mdb"));
}

```

接着输入以下代码：

```

/// <summary>
/// 获得所有的客户编号
/// </summary>
/// <returns>由客户编号字符串组成的数组</returns>
[WebMethod]
public string[] GetCustomerID()
{
    List<string> result = new List<string>();
    using (IDbConnection conn = CreateConnection())
    {
        conn.Open();
        using (IDbCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText = "Select CustomerID From Customers";
            IDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                if (reader.IsDBNull(0) == false)
                {
                    result.Add(reader.GetString(0));
                }
            }
            reader.Close();
        }
    }
    return result.ToArray();
}

```

对 `GetCustomerID` 方法，定义的时候并没有什么特别的，只不过它附加了一个名为“`WebMethod`”的特性，该特性类型全名为 `System.Web.Services.WebMethodAttribute`。

在 `System.Web.Services.WebService` 的派生类型中定义的公开方法，只要附加上 `WebMethodAttribute` 特性就成为一个对外公开发布的 Web 方法。其他应用系统可以通过 Web Service 机制调用这些 Web 方法。

为了容纳一个客户的基本信息，需要新建一个名为 `CustomerInfo` 的类型，定义该类型的代码如下：

```

/// <summary>
/// 客户信息对象
/// </summary>
public class CustomerInfo
{
    private string _CustomerID = null;
    /// <summary>
    /// 客户编号
    /// </summary>
    public string CustomerID
    {
        get { return _CustomerID; }
    }
}

```

```
        set { _CustomerID = value; }
    }

    private string _CompanyName = null;
    /// <summary>
    /// 公司名称
    /// </summary>
    public string CompanyName
    {
        get { return _CompanyName; }
        set { _CompanyName = value; }
    }

    private string _ContactName = null;
    /// <summary>
    /// 联系人
    /// </summary>
    public string ContactName
    {
        get { return _ContactName; }
        set { _ContactName = value; }
    }

    private string _ContactTitle = null;
    /// <summary>
    /// 联系人职务
    /// </summary>
    public string ContactTitle
    {
        get { return _ContactTitle; }
        set { _ContactTitle = value; }
    }

    private string _Address = null;
    /// <summary>
    /// 地址
    /// </summary>
    public string Address
    {
        get { return _Address; }
        set { _Address = value; }
    }

    private string _City = null;
    /// <summary>
    /// 城市
    /// </summary>
    public string City
    {
        get { return _City; }
        set { _City = value; }
    }

    private string _Region = null;
    /// <summary>
    /// 地区
    /// </summary>
    public string Region
    {
```

```

        get { return _Region; }
        set { _Region = value; }
    }

    private string _PostalCode = null;
    /// <summary>
    /// 邮政编码
    /// </summary>
    public string PostalCode
    {
        get { return _PostalCode; }
        set { _PostalCode = value; }
    }

    private string _Country = null;
    /// <summary>
    /// 城市
    /// </summary>
    public string Country
    {
        get { return _Country; }
        set { _Country = value; }
    }

    private string _Phone = null;
    /// <summary>
    /// 电话
    /// </summary>
    public string Phone
    {
        get { return _Phone; }
        set { _Phone = value; }
    }

    private string _Fax = null;
    /// <summary>
    /// 传真
    /// </summary>
    public string Fax
    {
        get { return _Fax; }
        set { _Fax = value; }
    }

    private string _Email = null;
    /// <summary>
    /// 电子信箱
    /// </summary>
    public string Email
    {
        get { return _Email; }
        set { _Email = value; }
    }

    private string _WebSite = null;
    /// <summary>
    /// 网址
    /// </summary>
    public string WebSite

```

```
    {  
        get { return _WebSite; }  
        set { _WebSite = value; }  
    }  
}
```

然后在 **CustomerService** 类型中需要定义一个名为 **GetCustomer** 的方法，其代码如下：

```
/// <summary>  
/// 根据编号获得完整的客户信息  
/// </summary>  
/// <param name="id">编号</param>  
/// <returns>客户信息对象</returns>  
[WebMethod]  
public CustomerInfo GetCustomer(string id)  
{  
    if (id == null || id.Trim().Length == 0)  
    {  
        return null;  
    }  
    CustomerInfo result = null;  
    using (IDbConnection conn = CreateConnection())  
    {  
        conn.Open();  
        using (IDbCommand cmd = conn.CreateCommand())  
        {  
            cmd.CommandText = @"  
Select  
    CustomerID ,  
    CompanyName ,  
    ContactName ,  
    ContactTitle ,  
    Address,  
    City ,  
    Region,  
    PostalCode ,  
    Country ,  
    Phone,  
    Fax ,  
    Email,  
    WebSite  
From Customers  
Where CustomerID = ?";  
            IDbDataParameter p = cmd.CreateParameter();  
            p.Value = id;  
            cmd.Parameters.Add(p);  
            IDataReader reader = cmd.ExecuteReader(CommandBehavior.SingleRow);  
            if (reader.Read())  
            {  
                result = new CustomerInfo();  
                if (reader.IsDBNull(0) == false)  
                {  
                    result.CustomerID = reader.GetString(0);  
                }  
                if (reader.IsDBNull(1) == false)  
                {  
                    result.CompanyName = reader.GetString(1);  
                }  
                if (reader.IsDBNull(2) == false)  
                {
```

```

        result.ContactName = reader.GetString(2);
    }
    if (reader.IsDBNull(3) == false)
    {
        result.ContactTitle = reader.GetString(3);
    }
    if (reader.IsDBNull(4) == false)
    {
        result.Address = reader.GetString(4);
    }
    if (reader.IsDBNull(5) == false)
    {
        result.City = reader.GetString(5);
    }
    if (reader.IsDBNull(6) == false)
    {
        result.Region = reader.GetString(6);
    }
    if (reader.IsDBNull(7) == false)
    {
        result.PostalCode = reader.GetString(7);
    }
    if (reader.IsDBNull(8) == false)
    {
        result.Country = reader.GetString(8);
    }
    if (reader.IsDBNull(9) == false)
    {
        result.Phone = reader.GetString(9);
    }
    if (reader.IsDBNull(10) == false)
    {
        result.Fax = reader.GetString(10);
    }
    if (reader.IsDBNull(11) == false)
    {
        result.Email = reader.GetString(11);
    }
    if (reader.IsDBNull(12) == false)
    {
        result.WebSite = reader.GetString(12);
    }
    } //if
    reader.Close();
} //using
} //using
return result;
}

```

这个方法的参数为一个字符串，用于指定客户信息的编号，方法返回值为一个 `CustomerInfo` 类型的对象实例。方法附加了 `WebMethod` 特性被标注为一个对外公开的 Web 方法。

到此读取客户信息使用的 Web Service 开发完毕。

从开发过程来看，在 ASP.NET 中，Web Service 实际上是一种特殊的页面，只是这种页面没有用户界面，没有使用 HTML 代码和 Web 控件，有的只是一个个对外公开的 Web 方法。

10.5 发布 Web Service

Web Service 应用程序实际上就是一种 ASP.NET 应用程序，因此其发布与 ASP.NET 应用程序一样，在 IIS 中添加虚拟目录，设置一些权限即可。

打开 IE 浏览器，输入访问 CustomerService.asmx 页面的 URL 地址，若浏览器显示的内容如图 10-5 所示，则表明 Web Service 发布成功。



图 10-5 IE 浏览器

10.6 使用 Web Service

Web Service 开发后需要被其他应用系统使用。使用 C#，我们可以很方便地在 Windows 应用程序或 ASP.NET 应用程序中使用 Web Service。

10.7 在 Windows 应用程序中使用 Web Service

在 VS.NET 中新建一个名为“第一个 WebService 应用程序_WinForm 客户端”的 C# Windows 应用程序项目。


10.7.1 添加 Web 引用

要使用 Web Service，首先需要添加对 Web Service 的引用。

在 VS.NET 中，单击主菜单项目“项目→添加 Web 引用”，弹出如图 10-6 所示的对话框。



图 10-6 “添加 Web 引用”对话框

在该对话框中的 URL 地址栏中输入 CustoomrService.asmx 页面的完整 URL 地址，单击后面的绿色箭头小按钮“”，则对话框右边会提示“找到 1 个服务：-CustomerService”，而 Web 引用名默认设置为服务器名称，在图 10-6 中为“localhost”，用户也可以修改这个 Web 引用名。

单击“添加引用”后，系统会关闭对话框。在解决方案资源管理器的树状列表中将显示出新的 Web 引用项目，如图 10-7 所示。



图 10-7 树状列表

10.7.2 使用 Web Service

新建一个名为“frmWebServiceClient”的窗体，窗体的用户界面的设计如图 10-8 所示。

图 10-8 最上面的 Web 服务地址输入框命名为“txtWebServiceURL”，用于输入 Web Service 的 URL 地址；客户编号列表控件命名为“lstCustomerID”，用于列出所有的客户编号；而右边的文本框控件用于显示客户的详细信息。

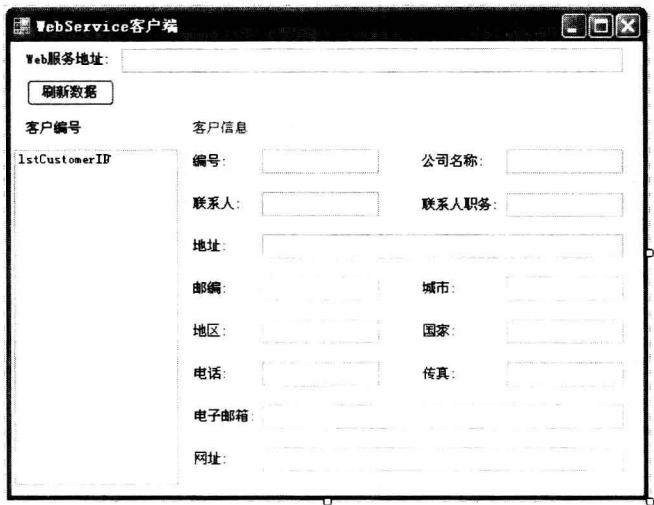


图 10-8 窗体设计器

在窗体设计器中双击“刷新数据”按钮，则显示出该按钮的点击事件处理方法的 C# 代码，输入以下点击事件处理代码。

```
/// <summary>
/// Web Service 代理对象
/// </summary>
private localhost.CustomerService myService
    = new localhost.CustomerService();
/// <summary>
/// 刷新按钮点击事件处理
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnRefreshView_Click(object sender, EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;
    try
    {
        lstCustomerID.Items.Clear();
        //设置 Web Service 的 URL 地址
        myService.Url = this.txtWebServiceURL.Text;
        //读取所有的客户编号
        string[] ids = myService.GetCustomerID();
        //将编号添加到列表中
        lstCustomerID.Items.AddRange(ids);
    }
    catch (Exception ext)
    {
        MessageBox.Show(ext.Message);
    }
    this.Cursor = Cursors.Default;
}
```

在这段代码中，定义了一个类型为 localhost.CustomerService 名为 myService 的全局变量。
类型 localhost.CustomerService 是添加 Web 引用后，VS.NET 根据 Web Service 的配置信息自

动生成的 C#源代码所生成的类型,用于代理 WebService 服务器端的 CustomerService 类型。
localhost 就是 Web 引用的名称, CustomerService 就是 Web Service 对象的名称。

可以查到定义 localhost.CustomerService 类型的代码如下:

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.Web.Services",
"4.0.30319.1")]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="CustomerServiceSoap",
Namespace="http://tempuri.org/")]
public partial class CustomerService :
    System.Web.Services.Protocols.SoapHttpClientProtocol {

    public CustomerService() {
        this.Url = global::第一个WebService 应用程序_WinForm 客户端.Properties.
Settings.Default.第一个WebService 应用程序_WinForm 客户端_localhost_Customer-
Service;
        if ((this.IsLocalFileSystemWebService(this.Url) == true)) {
            this.UseDefaultCredentials = true;
            this.useDefaultCredentialsSetExplicitly = false;
        }
        else {
            this.useDefaultCredentialsSetExplicitly = true;
        }
    }

    public new string Url {
        get {
            return base.Url;
        }
        set {
            if (((this.IsLocalFileSystemWebService(base.Url) == true)
                && (this.useDefaultCredentialsSetExplicitly == false))
                && (this.IsLocalFileSystemWebService(value) == false))) {
                base.UseDefaultCredentials = false;
            }
            base.Url = value;
        }
    }

    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.
org/GetCustomerID", RequestNamespace="http://tempuri.org/",
ResponseNamespace="http://tempuri.org/",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public string[] GetCustomerID() {
        object[] results = this.Invoke("GetCustomerID", new object[0]);
        return ((string[])(results[0]));
    }

    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http:
//tempuri.org/GetCustomer", RequestNamespace="http://tempuri.org/", Response-
Namespace="http://tempuri.org/", Use=System.Web.Services.Description.
SoapBindingUse.Literal, ParameterStyle=System.Web.Services.Protocols.
SoapParameterStyle.Wrapped)]
    public CustomerInfo GetCustomer(string id) {
        object[] results = this.Invoke("GetCustomer", new object[] {
            id});
    }
}
```

```
        return ((CustomerInfo)(results[0]));
    }

    private bool IsLocalFileSystemWebService(string url) {
        if ((url == null)
            || (url == string.Empty)) {
            return false;
        }
        System.Uri wsUri = new System.Uri(url);
        if ((wsUri.Port >= 1024)
            && (string.Compare(wsUri.Host, "localhost",
                System.String Comparison.OrdinalIgnoreCase) == 0)) {
            return true;
        }
        return false;
    }

    ----- 定义了其他类型成员 -----
}
```

客户端使用的 `CustomerService` 代理类型派生自类型 `System.Web.Services.Protocols.SoapHttpClientProtocol`，它提供了对服务器端 `CustomerService` 类型所有 Web 方法的映射，因此通过调用它的 `GetCustomerID` 方法能转而调用服务器端的 `CustomerService` 类型的 `GetCustomerID` 方法，来获得系统中所有客户的编号。

为 `lstCustomerID` 控件添加以下代码，以便处理 `SelectedIndexChanged` 事件。

```
localhost.CustomerInfo info = null;
if (lstCustomerID.Text != null && lstCustomerID.Text.Trim().Length > 0)
{
    try
    {
        info = myService.GetCustomer(lstCustomerID.Text);
    }
    catch (Exception ext)
    {
        MessageBox.Show(ext.Message);
    }
}
if (info == null)
{
    txtAddress.Text = "";
    txtCity.Text = "";
    txtCustomerID.Text = "";
    txtCompanyName.Text = "";
    txtContactName.Text = "";
    txtContactTitle.Text = "";
    txtCountry.Text = "";
    txtEmail.Text = "";
    txtFax.Text = "";
    txtPhone.Text = "";
    txtPostalCode.Text = "";
    txtRegion.Text = "";
    txtWebSite.Text = "";
}
else
{
    txtAddress.Text = info.Address;
    txtCity.Text = info.City;
```

```

txtCustomerID.Text = info.CustomerID;
txtCompanyName.Text = info.CompanyName;
txtContactName.Text = info.ContactName;
txtContactTitle.Text = info.ContactTitle;
txtCountry.Text = info.Country;
txtEmail.Text = info.Email;
txtFax.Text = info.Fax;
txtPhone.Text = info.Phone;
txtPostalCode.Text = info.PostalCode;
txtRegion.Text = info.Region;
txtWebSite.Text = info.WebSite;
}

```

在这段代码中，程序调用 `myService` 对象的 `GetCustomer` 方法获得一个 `localhost.CustomerInfo` 类型的对象实例，然后将对象的属性值填充到用户界面中的文本框控件中。

`localhost.CustomerInfo` 类型也是在添加 Web 引用过程中，VS.NET 自动生成的 C# 代码所形成的类型。可以查到定义该类型的 C# 代码为：

```

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.Xml", "4.0.30319.1")]
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://tempuri.org/")]
public partial class CustomerInfo {

    private string customerIDField;

    private string companyNameField;

    /// <remarks/>
    public string CustomerID {
        get {
            return this.customerIDField;
        }
        set {
            this.customerIDField = value;
        }
    }

    /// <remarks/>
    public string CompanyName {
        get {
            return this.companyNameField;
        }
        set {
            this.companyNameField = value;
        }
    }

    ----- 定义了其他类型成员 -----
}

```

从这段代码可以看到，客户端的 `localhost.CusotmerInfo` 类型提供了对服务器端的 `CustomerInfo` 类型的所有公开属性的映射。服务器端 `CustomerInfo` 对象的所有公开属性值都会被设置到客户端的 `localhost.CustomerInfo` 的公开属性。

到此，使用 Web Service 的客户端 Windows 应用程序开发完毕，该程序运行界面如图 10-9 所示。



图 10-9 WebService 客户端

在图 10-9 中，输入 Web Service 的 URL 地址，然后单击“刷新数据”就能在左边的列表中显示出所有的客户编号，单击客户编号列表中的某个编号，右边就能显示该编号客户的详细信息。

10.8 在 ASP.NET 应用程序中使用 Web Service

在 VS.NET 中新建一个名为“第一个 WebService 应用程序 ASPNET 客户端”的 C# ASP.NET 工程。

单击 VS.NET 主菜单项目“项目→添加 Web 引用”，弹出如图 10-10 所示的对话框。



图 10-10 “添加 Web 引用”对话框

在该对话框中单击“添加引用”按钮，即可将 Web Service 引入到本 ASP.NET 项目中。
打开主页面 Default.aspx 的页面编辑器，其页面设计如图 10-11 所示。

body

Web服务地址:

刷新数据

客户编号

未绑定

客户信息

编号:

公司名称:

联系人:

联系人职务:

地址:

邮编:

城市:

地区:

国家:

电话:

传真:

电子邮箱:

网址:

图 10-11 页面设计

页面中各个控件的名称和 10.7 节中介绍的 Windows 窗体中各个控件的名称一样。设置左边的列表框控件的 AutoPostBack 属性值为 True。

双击“刷新数据”按钮，会显示该按钮的点击事件处理 C#方法代码，在该 C#方法体中输入以下代码：

```
try
{
    using (localhost.CustomerService myService
        = new localhost.CustomerService())
    {
        //设置 Web Service 的 URL 地址
        myService.Url = this.txtWebServiceURL.Text;
        lstCustomerID.Items.Clear();
        //读取所有的客户编号
        string[] ids = myService.GetCustomerID();
        //将编号添加到列表中
        foreach (string id in ids)
        {
            lstCustomerID.Items.Add(id);
        }
    }
}
catch (Exception ext)
{
    this.ClientScript.RegisterStartupScript(
        this.GetType(),
        "RefreshView",
        "alert('" + ext.Message + "');",
        true);
}
```

这段代码和 Windows 演示程序的“刷新数据”按钮的点击事件处理代码很相似，原理也一样。创建一个 CustomerService 代理对象，设置它的 URL 属性值，然后调用它的 GetCustomerID

方法获得所有的客户编号数组，最后填充到列表中。

接着为客户编号列表框的 **SelectedIndexChanged** 事件添加以下处理代码：

```
localhost.CustomerInfo info = null;
using (localhost.CustomerService myService
    = new localhost.CustomerService())
{
    //设置 Web Service 的 URL 地址
    myService.Url = this.txtWebServiceURL.Text;

    if (lstCustomerID.Text != null && lstCustomerID.Text.Trim().Length >
0)
    {
        try
        {
            info = myService.GetCustomer(lstCustomerID.Text);
        }
        catch (Exception ext)
        {
            this.ClientScript.RegisterStartupScript(
                this.GetType(),
                "RefreshView",
                "alert('" + ext.Message + "');",
                true);
        }
    }
}
if (info == null)
{
    txtAddress.Text = "";
    txtCity.Text = "";
    txtCustomerID.Text = "";
    txtCompanyName.Text = "";
    txtContactName.Text = "";
    txtContactTitle.Text = "";
    txtCountry.Text = "";
    txtEmail.Text = "";
    txtFax.Text = "";
    txtPhone.Text = "";
    txtPostalCode.Text = "";
    txtRegion.Text = "";
    txtWebSite.Text = "";
}
else
{
    txtAddress.Text = info.Address;
    txtCity.Text = info.City;
    txtCustomerID.Text = info.CustomerID;
    txtCompanyName.Text = info.CompanyName;
    txtContactName.Text = info.ContactName;
    txtContactTitle.Text = info.ContactTitle;
    txtCountry.Text = info.Country;
    txtEmail.Text = info.Email;
    txtFax.Text = info.Fax;
    txtPhone.Text = info.Phone;
    txtPostalCode.Text = info.PostalCode;
    txtRegion.Text = info.Region;
    txtWebSite.Text = info.WebSite;
}
```

在这里，程序创建 `CustomerService` 对象，根据列表框中当前客户的编号调用 `CustomerService` 对象的 `GetCustomer` 方法，获得一个 `CustomerInfo` 对象，然后将其属性值填充到页面右边的各个文本框中。

10.9 Web Service 原理

通过上面开发的 Web Service 服务器端程序、Web Service WinForm 客户端和 ASP.NET 客户端程序，我们可以发现使用 VS.NET 开发和调用 Web Service 是非常方便的。

Web Service 可以说是 Web 页面浏览和本地调用的结合。

图 10-12 是 Web 页面浏览的原理图。

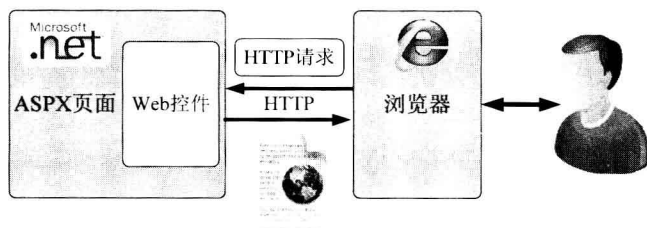


图 10-12 Web 页面浏览原理图

在图 10-12 中，用户操作浏览器向某个 ASPX 页面发起 HTTP 请求，向其传递一些地址参数或表单数据；ASPX 页面包含若干 Web 控件，接收输入的参数，经过一番处理生成 HTML 文档，然后作为返回值传递给客户端浏览器；客户端浏览器接收 HTML 文档并展现给用户。

图 10-13 是应用程序调用软件组件的原理图。

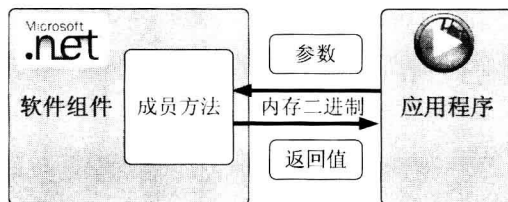


图 10-13 应用程序调用软件组件原理图

其实这里的软件组件就是对象实例，它包含若干个成员方法，应用程序将输入参数直接传递给软件组件的成员方法；成员方法接收输入的参数，经过一番处理生成返回值，然后直接传递给应用程序，应用程序就能接收返回值而知晓执行结果了。

图 10-14 是 Web Service 的原理图。

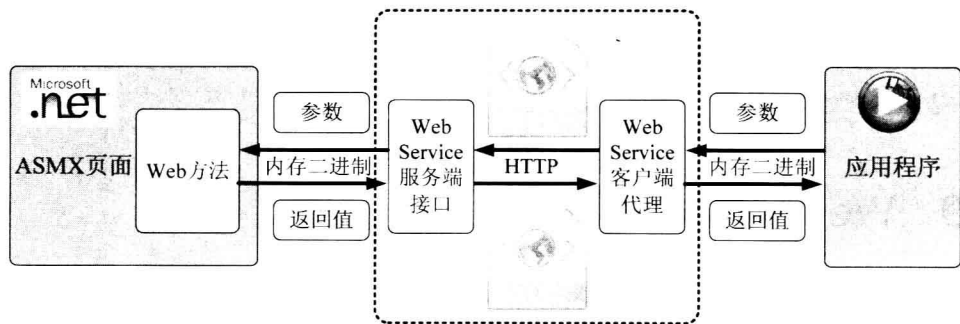


图 10-14 Web Service 原理图

整个过程的具体步骤如下：

- (1) 应用程序向 Web 方法发起调用请求，将准备好的输入参数发往 Web Service 客户端代理，然后等待。
- (2) Web Service 客户端代理接收参数，通过 SOAP 序列化操作将它们转换为一个 XML 文档，接着向 ASMX 页面发起 HTTP 请求，参数 XML 文档就作为表单数据发送过去，然后等待 HTTP 响应。
- (3) Web Service 服务器端接口接收 HTTP 请求数据，将其中的参数 XML 文档进行反序列化操作，还原出原始的参数值，然后转而调用 Web Service 对象中的 Web 方法，等待 Web 方法的处理。
- (4) Web Service 对象中的 Web 方法体接收输入的参数，执行一番操作，然后生成返回值传递给 Web Service 服务端接口。
- (5) Web Service 服务端接口接收 Web 方法的返回值，使用 SOAP 序列化操作将其转换为一个 XML 文档，然后作为 HTTP 响应数据通过网络传递到 Web Service 客户端代理。
- (6) Web Service 客户端代理接收 XML 文档，使用 SOAP 反序列化操作还原出原始的返回值，然后返回给应用程序。
- (7) 应用程序获得返回值后进行后续处理。

从应用程序的角度看，Web Service 客户端代理对象就是远程服务器上的 Web Service 对象的一个影子。虽然 Web Service 对象远在天边，但通过代理对象，它好像就在跟前为应用程序做贴身服务，而应用程序调用 Web Service 对象的功能和调用其他本地对象的功能一样好使。

从 Web Service 对象的角度看，Web Service 服务器端接口就是远程客户端应用程序的代理，通过这个接口，远程的应用程序就像在跟前调用自己的功能模块一样。

在图 10-14 中，虚线框部分对于 Web Service 服务对象和客户端应用程序来说是透明的，从便于理解的角度看，可以虚化掉而形成本地方法调用的模式，这样便实现了分布式软件架构和本地软件架构的统一。

分析 Web Service 的执行过程，可以知道 XML 是 Web Service 的技术基础，因此 Web Service 也被称为 XML Web Service。Web 方法调用过程中的输入参数和返回值都会在原始数据和 XML 文档之间进行转换，这个技术称为 SOAP 技术。SOAP 序列化/反序列实际上就是一种 XML 序列化/反序列化，因此开发和调用复杂的 Web Service 技术是需要知道一些 XML 序列化/反序列化知识的。

开发第一个 ADO.NET 数据库应用程序

ADO.NET 是 .NET 框架中很大的一部分，专门用于进行数据的访问，是开发人员经常用到的模块。

11.1 ADO.NET 数据库访问概述

对于初学者来说，ADO.NET 的框架图如图 11-1 所示。在 ADO.NET 框架中，ADO.NET 主要有两个功能块，一个是数据集，另一个是数据提供者。

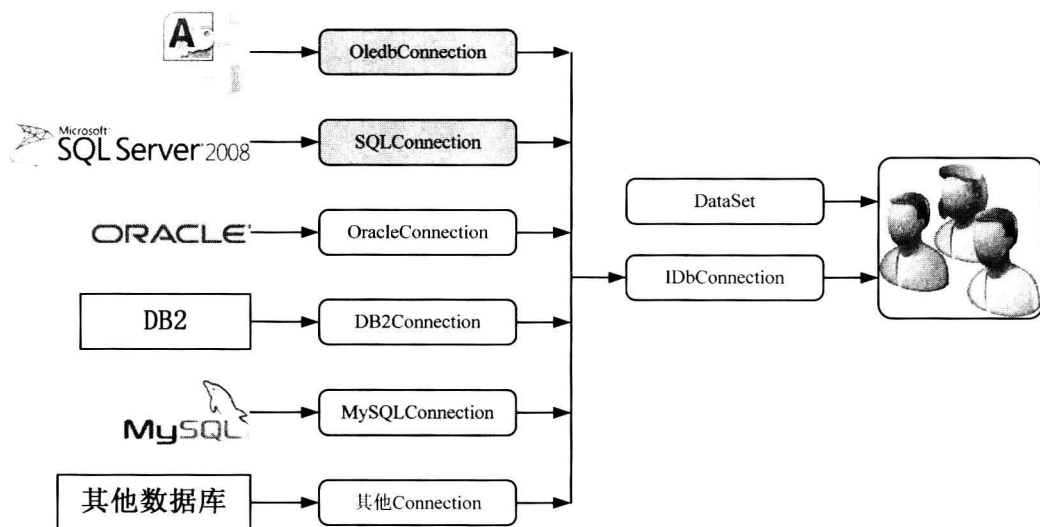


图 11-1 ADO.NET 框架图

数据集模块是以 `System.Data.DataSet` 类型为核心的，包括 `System.Data.DataTable`、`System.Data.DataColumn`、`System.Data.DataRow`、`System.Data.DataRelation` 等类型，用于维护在内存中的数据集合。这个数据集合既可以当做数据库记录在内存中的缓存，也可以作为内存中的

小型数据库，还可以作为与其他系统交互数据的容器。

数据提供者是一个以 `System.Data.IDbConnection` 接口为核心的，还包括 `System.Data.IDbCommand`、`System.Data.IDataReader` 等接口类型，定义了一种通用的数据提供者接口机制。在这种机制的基础上，ADO.NET 包含很多针对特定类型的数据库的数据提供者。比如针对 Access、MS SQL Server、Oracle、DB2、MySQL 甚至用户自定义的数据库，都可以从 `IDbConnection` 接口上实现相应的接口。而开发者使用统一的 `IDbConnection` 接口即可使用诸多特定类型的数据提供者，简化了编程，并提供了很强的灵活性和可扩展性。

11.2 建立 C# 应用程序项目

在 VS.NET 中新建一个名为“第一个 ADONET 应用程序”的 ASP.NET 应用程序，然后将演示数据库 `Customers.mdb` 复制到工程目录下的 `App_Data` 子目录下。

数据库 `Customers.mdb` 中有一个数据表 `Customers`，该数据表存储了客户的基本信息。表 11-1 列出了 `Customer` 数据表中的部分内容，此处没有列出所有的记录和所有的字段。

表 11-1 Customer 数据表中的部分内容

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
1	少室山公司	方证	采购员	东园西甲 30 号	长平
2	擎天航空	雷震子	销售代表	常保阁东 80 号	莫斯科
3	华夏工程	李大禹	市场经理	广发北路 10 号	幽州
4	武当投资	宋青书	物主	临翠大街 80 号	巴伐利亚
5	擎天南京公司	大星星	物主	花园东街 90 号	许安

11.2.1 快速读取数据

由于要有多个页面连接数据库访问数据，因此新建一个名为 `Util` 的类型，并在该类型中使用以下代码定义一个静态函数。

```
/// <summary>
/// 创建一个数据库连接对象
/// </summary>
/// <returns></returns>
public static IDbConnection CreateConnection()
{
    return new OleDbConnection(
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + System.Web.HttpContext.Current.Server.MapPath(
            "App_Data\\Customers.mdb"));
}
```

这个函数能创建一个数据库连接对象，并设置正确的数据库连接字符串。这样其他页面中就能直接调用这个方法来获得一个数据库连接对象。

使用类型 `System.Data.IDataReader` 能快速地读取数据库中的数据，但只能顺序地读，不能回头读以前的数据，也不能修改数据。

新建一个名为 “PageUseDataReader.aspx” 的 Web 页面，打开该页面的 C# 代码。输入以下代码定义一个名为 `TableHtml` 的全局变量。

```
public string TableHtml = null;
```

在该页面的 `Page_Load` 方法中输入以下代码：

```
using (IDbConnection conn = Util.CreateConnection())
{
    //打开数据库
    conn.Open();
    using (IDbCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = "Select * From Customers";
        IDataReader reader = cmd.ExecuteReader();

        StringBuilder htmlStr = new StringBuilder();
        htmlStr.Append("<table border='1'>");
        //输出表头
        htmlStr.AppendLine("<tr style='background-color:#cccccc'>");
        for (int iCount = 0; iCount < reader.FieldCount; iCount++)
        {
            htmlStr.AppendLine("<td>" + reader.GetName(iCount) + "</td>");
        }
        while (reader.Read())
        {
            //输出一行数据库记录
            htmlStr.AppendLine("<tr>");
            for (int iCount = 0; iCount < reader.FieldCount; iCount++)
            {
                //判断当前记录值是否为空
                if (reader.IsDBNull(iCount))
                {
                    htmlStr.AppendLine("<td>[NULL]</td>");
                }
                else
                {
                    htmlStr.AppendLine("<td>"
                        + Convert.ToString( reader.GetValue( iCount ))
                        + "</td>");
                }
            }
            htmlStr.AppendLine("</tr>");
        }
        htmlStr.AppendLine("</table>");
        reader.Close();
        this.TableHtml = htmlStr.ToString();
    } //using
} //using
```

这段代码的作用就是调用 ADO.NET 读取数据并输出 HTML 文本内容。从这段代码可以看出，读取数据的主要过程为：

(1) 创建数据库连接对象。由于所有的数据库连接对象都实现了 `IDbConnection` 接口，因此使用 `IDbConnection` 类型的变量指向任何创建的数据库连接对象的实例。

(2) 设置数据库连接字符串。也就是设置 `IDbConnection` 类型的 `ConnectionString` 属性。数据库连接字符串一般采用“设置名称=值;设置名称=值”的样式。不同的数据库其连接字符串的内容格式是不一样的。

(3) 打开数据库连接。设置好数据库连接字符串后,就要打开数据库连接,也就是调用 `IDbConnection` 类型的 `Open` 方法。`Open` 方法没有返回值,若操作成功则一切正常,若方法内部发生错误则抛出异常。

(4) 创建命令对象。成功地连接数据库后,需要调用 `IDbConnection` 类型的 `CreateCommand` 方法创建一个 `IDbCommand` 类型的数据库命令对象。

(5) 准备查询数据库。通过设置 `IDbCommand` 对象的 `CommandText` 属性值来设置查询数据库使用的 SQL 命令文本。

(6) 查询数据库。执行 `IDbCommand` 对象的 `ExecuteReader` 方法来查询数据库,获得一个 `IDataReader` 对象。`IDataReader` 对象能以只读方式向前读取查询结果。

(7) 读取数据。就是开发者使用 `IDataReader` 对象的成员属性或方法来读取数据。此处使用 `FieldCount` 属性来获得查询所得的字段个数,使用 `GetName` 方法来获得字段名,使用 `GetValue` 来获得查询所得的字段值。

(8) 关闭数据库连接。读取数据完毕后,一定要记住关闭数据库,在此调用 `IDataReader` 对象的 `Close` 方法来关闭数据读取器,然后使用 `using` 语句隐式地调用 `IDbCommand` 对象和 `IDbConnection` 对象的 `Dispose` 方法来关闭数据库连接,释放所有资源。

可见,在 ADO.NET 中快速读取数据的过程就是“创建数据库连接对象→设置数据库连接字符串→打开数据库连接→创建数据库命令对象→设置 SQL 语句→查询数据获得数据读取器→读取数据→关闭数据读取器→关闭数据库连接”。

在该页面中,程序读取数据,然后拼凑出 HTML 代码,设置到 `TableHtml` 的全局变量中。

打开该页面的 HTML 代码文件,在 HTML 正文中插入代码“<%= TableHtml%>”,则该 ASPX 页面的代码如下:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="PageUseDataReader.
aspx.cs" Inherits="第一个 ADONET 应用程序.PageUseDataReader" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<%= TableHtml%>
</div>
</form>
</body>
</html>
```

ASP.NET 框架执行这个 ASPX 页面时，首先调用 C# 代码中的 Page_Load 方法，然后输出 ASPX 页面中的 HTML 代码。遇到代码 “<%= TableHtml%>”，则会解析为输出页面公开的成员属性或字段值，此处输出页面全局变量 TableHtml 的文本。本页面运行时输出的 HTML 代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>

</title></head>
<body>
    <form name="form1" method="post" action="UseDataReader.aspx" id="form1">
    <div>
        <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value= " /wEPDwUKLTgONTYxMzIxNWRk+yLGQTWea9pYsAs3R00Xa+G+9BQ=" />
    </div>

    <div>
        <table border='1'><tr style='background-color:#cccccc'>
<td>CustomerID</td>
<td>CompanyName</td>
<td>ContactName</td>
<td>ContactTitle</td>
<td>Address</td>
<td>City</td>
<td>Region</td>
<td>PostalCode</td>
<td>Country</td>
<td>Phone</td>
<td>Fax</td>
<td>Email</td>
<td>WebSite</td>
<tr>
<td>1</td>
<td>三川实业有限公司</td>
<td>刘小姐</td>
<td>销售代表</td>
<td>大崇明路 50 号</td>
<td>天津</td>
<td>华北</td>
<td>343567</td>
<td>中国</td>
<td>(030) 30074321</td>
<td>(030) 30765452</td>
<td>Silva@lisboncycle.com</td>
<td>http://www.microsoft.com.cn</td>
</tr>
<tr>----- 其他记录值 -----</tr>
</table>

    </div>
    </form>
</body>
</html>
```

页面的用户界面如图 11-2 所示。

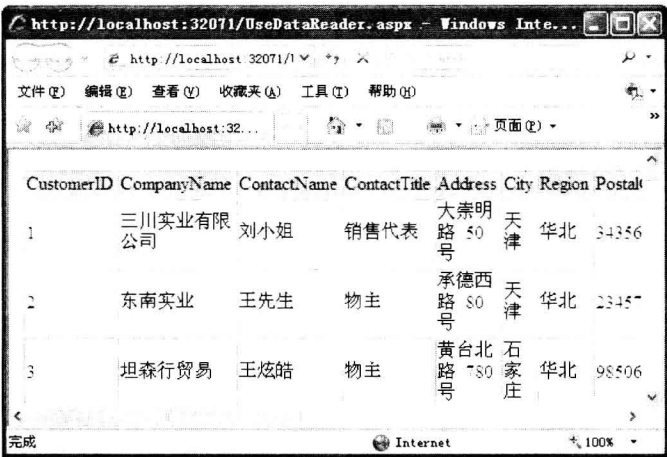


图 11-2 用户界面

11.2.2 数据源绑定

ASP.NET 中提供了数据源绑定功能，能在 Web 控件中直接显示 ADO.NET 查询所得的数据而不需要做多少编程工作。下面演示在 ASP.NET 中如何进行数据源绑定。

新建一个名为“PageUseDataBind.aspx”的页面，在这个页面中添加一个 GridView 控件，设置该控件的属性，如表 11-2 所示。

表 11-2 控件属性

属 性	功 能
ID	dvCustomers
AllowPaging	True，允许分页显示
BackColor	White
BorderColor	#3366CC
BorderStyle	None
BorderWidth	1px
CellPadding	4
FooterStyle.BackColor	#99CCCC，分页导航条背景色
FooterStyle.ForeColor	#003399，分页导航条文本颜色
HeaderStyle.BackColor	#003399，标题行背景色
HeaderStyle.ForeColor	#CCCCFF，标题行文本颜色
RowStyle.BackColor	White，数据行背景色
RowStyle.ForeColor	#003399，数据行文本颜色
PageSize	8，每个分页显示的记录个数
PaperStyle.BackColor	#99CCCC，分页导航栏背景色
PaperStyle.ForeColor	#003399，分页导航栏文本颜色
Width	100%

该页面的用户界面的设计如图 11-3 所示。

在实践中开发人员可以使用更快捷的方式设置控件的外观样式。在界面设计器中选中 GridView 控件时，在控件属性编辑器中会出现“自动套用格式”功能链接，如图 11-4 所示。单击该链接，弹出如图 11-5 所示的对话框。

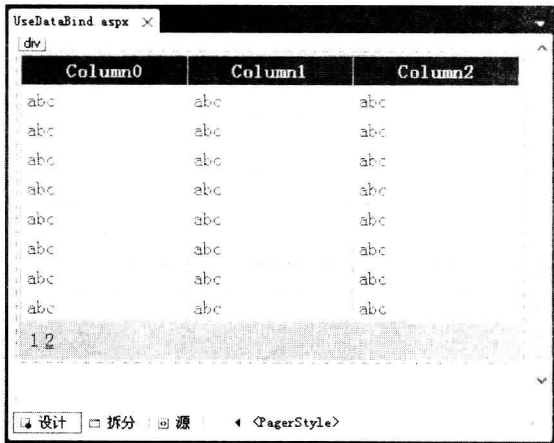


图 11-3 用户界面设计

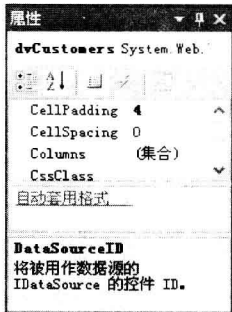


图 11-4 属性编辑器

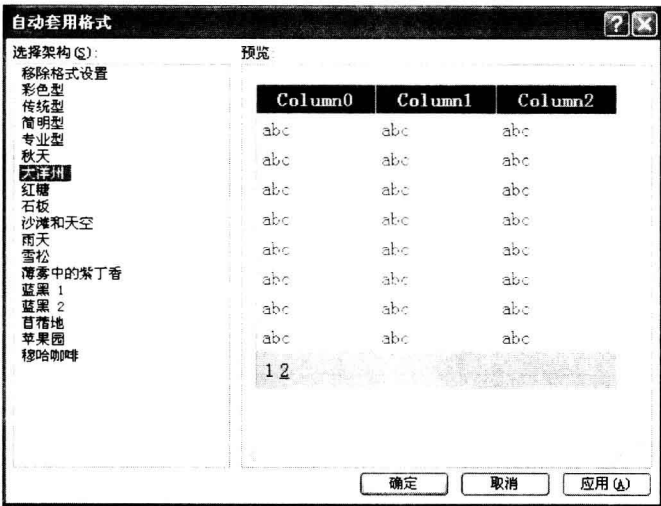


图 11-5 “自动套用格式”对话框

在该对话框中开发者可以为网格控件选择某个主题，然后单击“确定”或“应用”按钮，即可修改控件的各种外观样式的属性。

打开该页面的 C#代码，在其 Page_Load 方法中输入以下 C#代码：

```
using (IDbConnection conn = Util.CreateConnection())
{
    //打开数据库
```



```

conn.Open();
using (IDbCommand cmd = conn.CreateCommand())
{
    cmd.CommandText = "Select CustomerID , CompanyName , " +
        " ContactName , Address From Customers";
    IDataReader reader = cmd.ExecuteReader();
    DataTable table = new DataTable();
    table.Load(reader);
    reader.Close();
    //设置数据源
    this.dvCustomers.DataSource = table;
    //绑定数据源
    this.dvCustomers.DataBind();
}
}

```

在这段代码中，连接数据库，设置 SQL 查询文本，执行查询获得一个数据读取器，接着创建一个 `System.Data.DataTable` 对象实例，调用它的 `Load` 方法加载所有查询所得的数据到内存中；然后设置数据网格控件的 `DataSource` 属性，调用它的 `DataBind` 方法。

由于本控件还需要进行分页显示，因此需要处理该控件的 `PageIndexChanging` 事件。为该事件编写的代码如下：

```

protected void dvCustomers_PageIndexChanging(
    object sender,
    GridViewPageEventArgs e)
{
    this.dvCustomers.PageIndex = e.NewPageIndex;
    this.dvCustomers.DataBind();
}

```

这样 ADO.NET 产生数据源，ASP.NET 绑定数据源，两者协同工作就能为用户展现数据，该页面运行时的用户界面如图 11-6 所示。

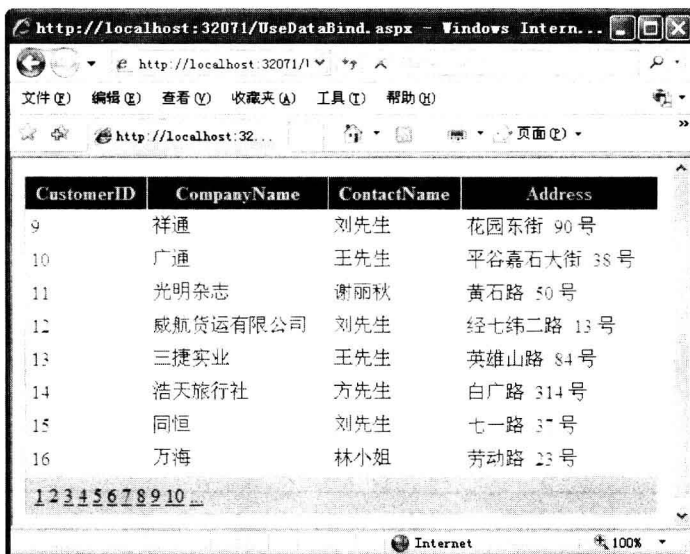


图 11-6 用户界面

在该用户界面中，用户可以单击最下面的分页标签来显示指定分页的数据。

可以看出，通过数据源绑定技术，开发者无须考虑如何生成显示数据使用的 HTML 代码，只需要对控件进行可视化设计，编程向控件提供数据源即可。这样能较大地降低开发工作量，提高软件功能的开发速度。

11.2.3 修改数据

使用 ADO.NET 能修改数据库中的数据，包括新增、修改和删除数据。

1. 客户记录列表页面

首先新建一个名为“PageCustomerList.aspx”的页面，在页面中放入一个“新增”按钮，该按钮的 HTML 代码为“<input type="button" value="新增" onclick="_InsertRecord();" />”。

然后在页面中放置一个名为“dvCustomers”的 GridView 控件。在界面设计器中选中这个控件，在属性编辑器中选中 Columns 属性，单击后面的小按钮，弹出如图 11-7 所示的“字段”对话框。

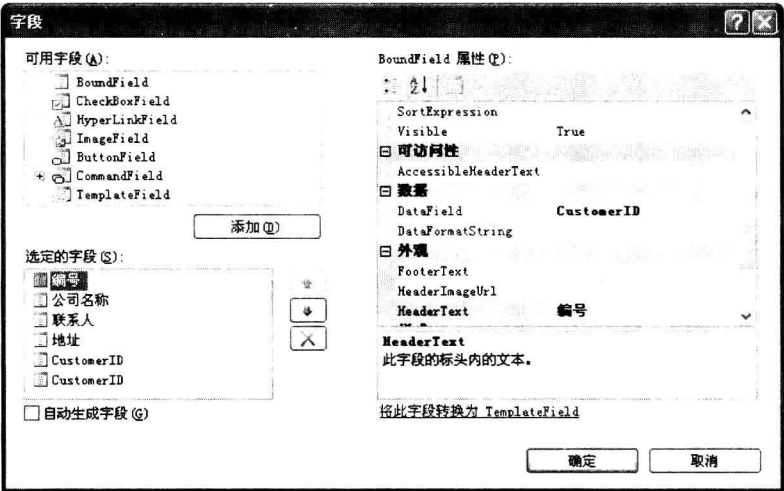


图 11-7 “字段”对话框

在该对话框中，取消“自动生成字段”的勾选状态，然后在可用字段列表中选择“BoundField”项目，单击“添加”按钮，向选定的字段列表中添加字段对象，并在右边设置各个字段对象的属性。各个字段的属性设置如表 11-3 所示。

表 11-3 字段属性

第一个字段:	
DataField	CustomerID
HeaderText	编号

续表

第二个字段:	
DataField	CompanyName
HeaderText	公司名称
第三个字段:	
DataField	ContactName
HeaderText	联系人
第四个字段:	
DataField	Address
HeaderText	地址
第五个字段:	
HtmlEncode	False, 该字段的内容不执行 HTML 编码, 原样输出
DataField	CustomerID
DataFormatString	<input type="button" value="修改" onclick="_UpdateRecord({0});">
第六个字段:	
HtmlEncode	False, 该字段的内容不执行 HTML 编码, 原样输出
DataField	CustomerID
DataFormatString	<input type="button" value="删除" onclick="_DeleteRecord(this , {0});">

经过上述设计该页面的设计界面如图 11-8 所示。



图 11-8 设计界面

该页面正文内容的 HTML 代码为:

```
客户列表
<input type="button" value="新增" onclick="_InsertRecord();" />
<br />
<asp:GridView ID="dvCustomers" runat="server" AllowPaging="True" BackColor=
"White"
```

```
BorderColor="#3366CC" BorderStyle="None" BorderWidth="1px" CellPadding=
"4" EnableModelValidation="True"
OnPageIndexChanging="dvCustomers_PageIndexChanging" PageSize="8"
Width="100%"
AutoGenerateColumns="False">
<Columns>
    <asp:BoundField DataField="CustomerID" HeaderText="编号" />
    <asp:BoundField DataField="CompanyName" HeaderText="公司名称" />
    <asp:BoundField DataField="ContactName" HeaderText="联系人" />
    <asp:BoundField DataField="Address" HeaderText="地址" />
    <asp:BoundField DataField="CustomerID" DataFormatString="&lt;input
type='button' value='修改' onclick='_UpdateRecord({0});'&gt;"
        HtmlEncode="False" />
    <asp:BoundField DataField="CustomerID" DataFormatString="&lt;input
type='button' value='删除' onclick='_DeleteRecord( this , {0});'&gt;"
        HtmlEncode="False" />
</Columns>
<FooterStyle BackColor="#99CCCC" ForeColor="#003399" />
<HeaderStyle BackColor="#003399" Font-Bold="True" ForeColor="#CCCCFF" />
<PagerStyle BackColor="#99CCCC" ForeColor="#003399" HorizontalAlign=
"Left" />
<RowStyle BackColor="White" ForeColor="#003399" />
<SelectedRowStyle BackColor="#009999" Font-Bold="True"
ForeColor="#CCFF99" />
</asp:GridView>
```

由于这里的按钮使用了 JavaScript 函数 `_InsertRecord`、`_UpdateRecord`、`_DeleteRecord`，因此还需使用以下 JavaScript 代码定义 `_InsertRecord` 函数。

```
//新增记录
function _InsertRecord() {
    if (window.showModalDialog(
        "PageEditCustomer.aspx?action=insert&flag=" + Math.random(),
        null,
        "dialogHeight:300px;dialogWidth:480px") == "InsertOK") {
        document.location.reload();
    }
}
```

在这段代码中，程序调用 `window.showModalDialog` 函数显示一个网页对话框，对话框内容为“`PageEditCustomer.aspx?action=insert&flag=某个随机数`”。若这个函数返回“`InsertOK`”值，则调用“`document.location.reload()`”来刷新这个页面，重新加载整个 HTML 文档。

在测试中，IE 浏览器显示网页对话框时很容易使用缓存的数据，因此使用没有实际意义的页面参数来生成不重复的页面地址，可以防止显示缓存的网页对话框中的内容。

使用以下 JavaScript 代码定义 `_UpdateRecord` 函数。

```
//修改记录
function _UpdateRecord(customerID) {
    if (window.showModalDialog(
        "PageEditCustomer.aspx?action=update&customerid="
        + customerID + "&flag=" + Math.random(),
        null,
        "dialogHeight:300px;dialogWidth:480px") == "UpdateOK") {
        document.location.reload();
    }
}
```

在这段代码中，程序显示一个网页对话框，若返回“UpdateOK”则刷新页面。本函数有一个参数，用于指明要处理的客户基本信息记录编号。

使用以下 JavaScript 代码定义 _DeleteRecord 函数。

```
//删除记录
function _DeleteRecord(sender, customerID) {
    if (window.showModalDialog(
        "PageEditCustomer.aspx?action=delete&customerid="
        + customerID + "&flag=" + Math.random(),
        null,
        "dialogHeight:300px;dialogWidth:480px") == "DeleteOK") {
        var row = sender;
        //删除按钮所在的表格行对象
        while (row != null) {
            if (row.tagName == "TR") {
                row.removeNode();
                break;
            }
            row = row.parentNode;
        }
    }
}
```

在这段代码中，程序显示一个网页对话框，若返回“DeleteOK”，则删除这条记录所在的 HTML 表格行对象。本函数第一个参数为调用该方法的 HTML 按钮对象，第二个参数用于指定要处理的客户基本信息记录编号。

打开 PageCustomerList.aspx 页面的 C#源代码，在 Page_Load 方法中输入以下代码：

```
using (IDbConnection conn = Util.CreateConnection())
{
    //设置数据库连接字符串
    conn.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + this.Server.MapPath("App_Data\\Customers.mdb");
    //打开数据库
    conn.Open();
    using (IDbCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = "Select CustomerID , CompanyName , " +
            " ContactName , Address From Customers Order By CompanyName";
        IDataReader reader = cmd.ExecuteReader();
        DataTable table = new DataTable();
        table.Load(reader);
        reader.Close();
        //设置数据源
        this.dvCustomers.DataSource = table;
        //绑定数据源
        this.dvCustomers.DataBind();
    } //using
} //using
```

处理数据表格控件的 PageIndexChanging 事件，其代码如下：

```
protected void dvCustomers_PageIndexChanging(
    object sender,
    GridViewPageEventArgs e)
```

```
{
    this.dvCustomers.PageIndex = e.NewPageIndex;
    this.dvCustomers.DataBind();
}
```

可以看出这个页面的 C# 代码和前面的 PageUseDataBind.aspx 一样。本页面运行时的用户界面如图 11-9 所示。

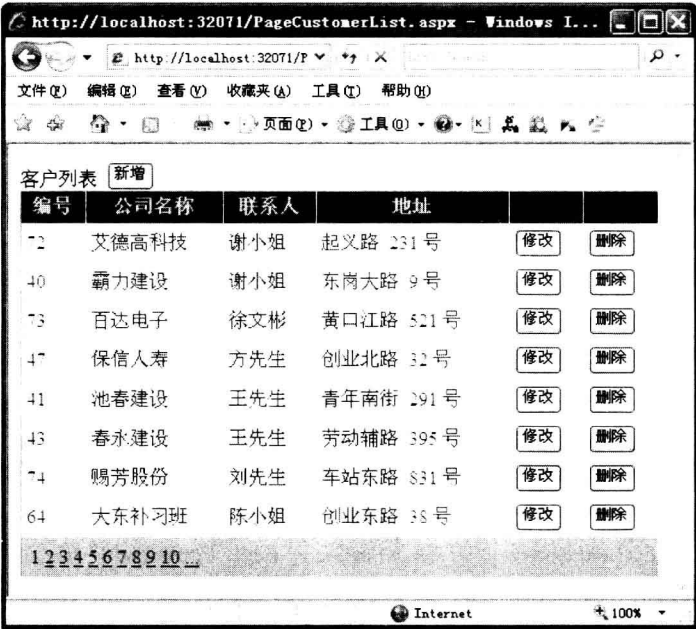


图 11-9 用户界面

2. 客户记录信息对象

本程序需要新增、修改和删除数据表 Customers 中的记录，为此定义一个客户记录信息对象来处理这些底层数据库操作，避免在上层的 ASPX 页面中出现大量和底层数据结构密切相关的代码。

新建一个名为 CustomerClass 的类型，该类型和数据表 Customers 相关联。对于该类型，首先需要定义和数据库字段一一对应的属性，为此输入以下代码：

```
private int _CustomerID = -1;
public virtual int CustomerID
{
    get { return _CustomerID; }
    set { _CustomerID = value; }
}

private string _CompanyName = null;

public virtual string CompanyName
{
    get { return _CompanyName; }
```

```
        set { _CompanyName = value; }
    }

    private string _ContactName = null;

    public virtual string ContactName
    {
        get { return _ContactName; }
        set { _ContactName = value; }
    }

    private string _ContactTitle = null;

    public virtual string ContactTitle
    {
        get { return _ContactTitle; }
        set { _ContactTitle = value; }
    }

    private string _Address = null;

    public virtual string Address
    {
        get { return _Address; }
        set { _Address = value; }
    }

    private string _City = null;

    public virtual string City
    {
        get { return _City; }
        set { _City = value; }
    }

    private string _Region = null;

    public virtual string Region
    {
        get { return _Region; }
        set { _Region = value; }
    }

    private string _PostalCode = null;

    public virtual string PostalCode
    {
        get { return _PostalCode; }
        set { _PostalCode = value; }
    }

    private string _Country = null;

    public virtual string Country
    {
        get { return _Country; }
        set { _Country = value; }
    }

    private string _Phone = null;
```

```
public virtual string Phone
{
    get { return _Phone; }
    set { _Phone = value; }
}

private string _Fax = null;

public virtual string Fax
{
    get { return _Fax; }
    set { _Fax = value; }
}

private string _Email = null;

public virtual string Email
{
    get { return _Email; }
    set { _Email = value; }
}

private string _WebSite = null;

public virtual string WebSite
{
    get { return _WebSite; }
    set { _WebSite = value; }
}
```

这样从数据表 Customers 中读取的字段值就可以一一设置到相对于的属性中。

接着使用以下代码定义一个成员 Read 方法。

```
public void Read(IDataReader reader)
{
    int index = reader.GetOrdinal("CustomerID");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.CustomerID = Convert.ToInt32(reader.GetValue(index));
    }
    index = reader.GetOrdinal("Address");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Address = Convert.ToString(reader.GetValue(index));
    }
    index = reader.GetOrdinal("City");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.City = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("CompanyName");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.CompanyName = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("ContactName");
    if (index >= 0 && reader.IsDBNull(index) == false)
```



```

    {
        this.ContactName = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("ContactTitle");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.ContactTitle = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("Country");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Country = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("Email");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Email = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("Fax");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Fax = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("Phone");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Phone = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("PostalCode");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.PostalCode = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("Region");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.Region = Convert.ToString(reader.GetValue(index));
    }

    index = reader.GetOrdinal("WebSite");
    if (index >= 0 && reader.IsDBNull(index) == false)
    {
        this.WebSite = Convert.ToString(reader.GetValue(index));
    }
}

```

在这段代码中，根据查询结果设置对象的属性值。

使用以下代码定义另外一个 **Read** 函数。

```

/// <summary>
/// 从数据库中读取一条客户信息记录
/// </summary>

```

```

/// <param name="connection">数据库连接对象</param>
/// <param name="customerID">指定的客户编号</param>
/// <returns>操作是否成功</returns>
public bool Read(IDbConnection connection, int customerID)
{
    using (IDbCommand cmd = connection.CreateCommand())
    {
        cmd.CommandText = @"
            Select
                CustomerID , CompanyName , ContactName , ContactTitle ,
                Address , City , Region , PostalCode ,
                Country , Phone , Fax , Email , WebSite
            From Customers
            Where CustomerID = @CustomerID";
        AddParameter(cmd, "CustomerID", customerID);
        IDataReader reader = cmd.ExecuteReader();
        bool result = false;
        if (reader.Read())
        {
            this.Read(reader);
            result = true;
        }
        reader.Close();
        return result;
    }
}

```

本方法的第一个参数为数据库连接对象，第二个参数为指定的客户记录编号。在这段代码中，程序创建数据库命令对象，设置 SQL 语句，添加 SQL 查询参数，然后执行查询获得一个数据读取器，调用上一个版本的 Read 方法读取数据，并返回结果。

这里使用添加 SQL 命令参数的方式来设置 SQL 命令对象，在实践中很多人会使用字符串拼凑的方式来简单地设置 SQL 命令对象，例如使用以下代码：

```

cmd.CommandText = @"
    Select
        CustomerID , CompanyName , ContactName , ContactTitle ,
        Address , City , Region , PostalCode ,
        Country , Phone , Fax , Email , WebSite
    From Customers
    Where CustomerID = " + customerID;

```

这是一种不安全的做法，会造成 SQL 注入式攻击的漏洞，因此不推荐使用，应该采用追加 SQL 命令参数的方式，并养成习惯。

这里调用了 AddParameter 方法，定义这个方法的代码如下：

```

private void AddParameter(IDbCommand cmd, string name, object Value)
{
    IDbDataParameter p = cmd.CreateParameter();
    p.ParameterName = name;
    p.Value = Value;
    cmd.Parameters.Add(p);
}

```

这个方法用于向一个 SQL 命令对象添加指定名称、指定数值的 SQL 命令参数对象。

本类型还定义了一个名为 Insert 的方法，用于向数据库中插入新的记录。定义该方法的 C#

代码如下:

```

/// <summary>
/// 插入数据
/// </summary>
/// <param name="connection">数据库连接对象</param>
/// <returns>操作是否成功</returns>
public bool Insert(IDbConnection connection)
{
    using (IDbCommand cmd = connection.CreateCommand())
    {
        cmd.CommandText = @"
            Insert Into Customers (
                CustomerID , CompanyName , ContactName , ContactTitle ,
                Address , City , Region , PostalCode , Country , Phone ,
                Fax , Email , WebSite )
            Values (
                @CustomerID , @CompanyName , @ContactName , @ContactTitle ,
                @Address , @City , @Region , @PostalCode , @Country , @Phone ,
                @Fax , @Email , @WebSite );";
        AddParameter( cmd , "CustomerID", this.CustomerID);
        AddParameter( cmd , "CompanyName", this.CompanyName);
        AddParameter( cmd , "ContactName", this.ContactName);
        AddParameter( cmd , "ContactTitle", this.ContactTitle);
        AddParameter( cmd , "Address", this.Address);
        AddParameter( cmd , "City", this.City);
        AddParameter( cmd , "Region", this.Region);
        AddParameter( cmd , "PostalCode", this.PostalCode);
        AddParameter( cmd , "Country", this.Country);
        AddParameter( cmd , "Phone", this.Phone);
        AddParameter( cmd , "Fax", this.Fax);
        AddParameter( cmd , "Email", this.Email);
        AddParameter( cmd , "WebSite", this.WebSite);
        int result = cmd.ExecuteNonQuery();
        return result != 0;
    }
}

```

这个方法的参数为数据库连接对象，在方法体中，程序创建数据库命令对象，设置 SQL 语句，添加 SQL 参数值，然后调用数据库命令对象的 `ExecuteNonQuery` 方法执行 SQL 语句。若操作成功，则 `ExecuteNonQuery` 的返回值一定不是零，因此可以据此判断操作是否成功。

本类型还定义一个 `Update` 方法，用于根据对象数据修改数据库中的数据。定义该方法的代码如下:

```

/// <summary>
/// 更新数据
/// </summary>
/// <param name="connection">数据库连接对象</param>
/// <returns>操作是否成功</returns>
public bool Update(IDbConnection connection)
{
    using (IDbCommand cmd = connection.CreateCommand())
    {
        cmd.CommandText = @"
            Update Customers Set
                CompanyName = @CompanyName ,
                ContactName = @ContactName ,

```

```
        ContactTitle = @ContactTitle ,
        Address = @Address ,
        City = @City ,
        Region = @Region ,
        PostalCode = @PostalCode ,
        Country = @Country ,
        Phone = @Phone ,
        Fax = @Fax ,
        Email = @Email ,
        WebSite = @WebSite
        Where CustomerID = @CustomerID";
    AddParameter( cmd , "CompanyName", this.CompanyName);
    AddParameter( cmd , "ContactName", this.ContactName);
    AddParameter( cmd , "ContactTitle", this.ContactTitle);
    AddParameter( cmd , "Address", this.Address);
    AddParameter( cmd , "City", this.City);
    AddParameter( cmd , "Region", this.Region);
    AddParameter( cmd , "PostalCode", this.PostalCode);
    AddParameter( cmd , "Country", this.Country);
    AddParameter( cmd , "Phone", this.Phone);
    AddParameter( cmd , "Fax", this.Fax);
    AddParameter( cmd , "Email", this.Email);
    AddParameter( cmd , "WebSite", this.WebSite);
    AddParameter( cmd , "CustomerID", this.CustomerID );
    int result = cmd.ExecuteNonQuery();
    return result != 0;
}
}
```

这个方法的参数是一个数据库连接对象，在这个方法内部，程序创建数据库命令对象，设置 SQL 语句，并根据属性值添加 SQL 命令参数对象，然后执行该 SQL 命令，返回执行结果。

本方法还定义了一个 Delete 方法，用于删除数据库记录，定义它的代码如下：

```
/// <summary>
/// 删除数据
/// </summary>
/// <param name="connection">数据库连接对象</param>
/// <returns>操作是否成功</returns>
public bool Delete(IDbConnection connection)
{
    using (IDbCommand cmd = connection.CreateCommand())
    {
        cmd.CommandText = "Delete From Customers Where CustomerID =
@CustomerID";
        AddParameter( cmd , "CustomerID", this.CustomerID );
        int result = cmd.ExecuteNonQuery();
        return result != 0;
    }
}
```

这个方法的参数为数据库连接对象，在本方法内部，程序创建一个数据库命令对象，设置 SQL 语句，添加 SQL 参数对象，然后执行该 SQL 命令，返回执行结果。

这个 CustomerClass 类型封装了直接操作数据库的底层功能，并向外提供调用接口，实现了用户界面层和数据库层之间的交接，形成了三层架构的中间层，在软件开发中应当树立这种思想。

此处 CustomerClass 类型和数据表 Customers 存在映射关系，很容易使用代码生成器自动生成它的源代码。

3. 客户详细信息编辑页面

从 PageCustomerList.aspx 的 HTML 代码中的 JavaScript 代码可以看出，系统还需要一个名为“PageEditCustomer.aspx”的页面，并接受名为“action”和“customerid”的页面的 URL 参数。action 参数值若等于 insert 则表示要新增记录，若等于 update 则表示要修改指定的记录，若等于 delete 则表示要删除指定的记录。customerid 页面参数用于指定要处理的客户基本信息编号。

为此新建一个名为“PageEditCustomer.aspx”的页面，并设计该页面的用户界面，如图 11-10 所示。

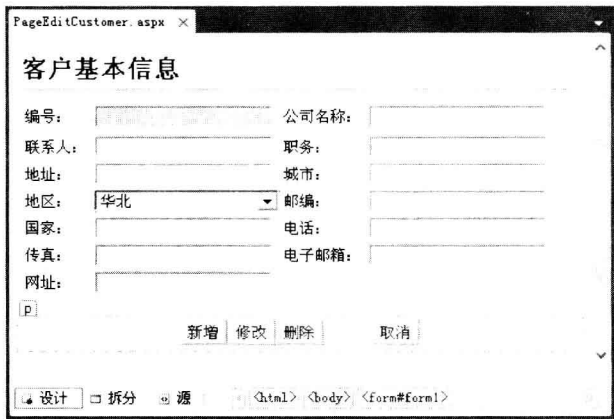


图 11-10 用户界面

该页面正文的 HTML 代码如下：

```
<form id="form1" runat="server" action="#">
<strong><span style="font-size: 16pt">客户基本信息</span></strong><br />
<hr />
<table>
  <tr>
    <td style="width: 54px; height: 26px;">
      <span style="font-size: 10pt">编号: </span>
    </td>
    <td style="width: 130px; height: 26px;">
      <asp:TextBox ID="txtCustomerID" runat="server" BackColor=
"#CCCCCC" ReadOnly="True"></asp:TextBox>
    </td>
    <td style="width: 67px; height: 26px;">
      <span style="font-size: 10pt">公司名称: </span>
    </td>
    <td style="width: 130px; height: 26px;">
      <asp:TextBox ID="txtCompanyName" runat="server"></asp:TextBox>
    </td>
  </tr>
</table>
```

```

        <td style="width: 54px">
            <span style="font-size: 10pt">联系人: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtContactName" runat="server"></asp:TextBox>
        </td>
        <td style="width: 67px">
            <span style="font-size: 10pt">职务: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtContactTitle" runat="server"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td style="width: 54px">
            <span style="font-size: 10pt">地址: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtAddress" runat="server"></asp:TextBox>
        </td>
        <td style="width: 67px">
            <span style="font-size: 10pt">城市: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtCity" runat="server"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td style="width: 54px">
            <span style="font-size: 10pt">地区: </span>
        </td>
        <td style="width: 130px">
            <asp:DropDownList ID="cboRegion" runat="server" Width="153px">
                <asp:ListItem>华北</asp:ListItem>
                <asp:ListItem>华南</asp:ListItem>
                <asp:ListItem>华东</asp:ListItem>
                <asp:ListItem>西南</asp:ListItem>
                <asp:ListItem>西北</asp:ListItem>
            </asp:DropDownList>
        </td>
        <td style="width: 67px">
            <span style="font-size: 10pt">邮编: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtPostalCode" runat="server"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td style="width: 54px">
            <span style="font-size: 10pt">国家: </span>
        </td>
        <td style="width: 130px">
            <asp:TextBox ID="txtCountry" runat="server"></asp:TextBox>
        </td>
        <td style="width: 67px">
            <span style="font-size: 10pt">电话: </span>
        </td>
        <td style="width: 130px">

```

```
<asp:TextBox ID="txtPhone" runat="server"></asp:TextBox>  
</td>  
</tr>  
<tr>  
    <td style="width: 54px">  
        <span style="font-size: 10pt">传真: </span>  
</td>  
    <td style="width: 130px">  
        <asp:TextBox ID="txtFax" runat="server"></asp:TextBox>  
</td>  
    <td style="width: 67px">  
        <span style="font-size: 10pt">电子邮箱: </span>  
</td>  
    <td style="width: 130px">  
        <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>  
</td>  
</tr>  
<tr>  
    <td style="width: 54px">  
        <span style="font-size: 10pt">网址: </span>  
</td>  
    <td style="width: 130px">  
        <asp:TextBox ID="txtWebSite" runat="server"></asp:TextBox>  
</td>  
    <td style="width: 67px">  
</td>  
    <td style="width: 130px">  
</td>  
</tr>  
</table>  
<p align="center">  
    <asp:Button ID="btnInsert" runat="server" Text="新增" Visible="False"  
OnClick="btnInsert_Click" />  
    <asp:Button ID="btnUpdate" runat="server" Text="修改" Visible="False"  
OnClick="btnUpdate_Click" />  
    <asp:Button ID="btnDelete" runat="server" Text="删除" Visible="False"  
OnClick="btnDelete_Click" />  
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
    <input type="button" value="取消" onclick="this.close();" />  
</p>  
</form>
```

在这个 HTML 代码中要注意的是，表单元素的 `action` 属性值为“#”，而默认是没有这个属性的，这表示将在本页面中进行发送处理；设置“新增”、“修改”、“删除”按钮的 `Visible` 属性值为“False”。

打开该页面的 C# 代码，在 Page Load 方法中输入以下代码：

```
if (this.IsPostBack == false)
{
    string action = this.Request.QueryString["action"];
    if (action == "insert")
    {
        this.Title = "新增客户信息";
        btnInsert.Visible = true;
        using (IDbConnection conn = Util.CreateConnection())
        {
            conn.Open();
```

```
        using (IDbCommand cmd = conn.CreateCommand())
        {
            cmd.CommandText =
                "Select Max( CustomerID ) + 1 From Customers";
            txtCustomerID.Text = cmd.ExecuteScalar().ToString();
        }
    }
}
else if (action == "update" || action == "delete")
{
    int customerID = Convert.ToInt32(
        this.Request.QueryString["customerid"]);
    using (IDbConnection conn = Util.CreateConnection())
    {
        conn.Open();
        CustomerClass instance = new CustomerClass();
        if (instance.Read(conn, customerID))
        {
            txtCustomerID.Text = instance.CustomerID.ToString();
            txtAddress.Text = instance.Address;
            txtCity.Text = instance.City;
            txtCompanyName.Text = instance.CompanyName;
            txtContactName.Text = instance.ContactName;
            txtContactTitle.Text = instance.ContactTitle;
            txtCountry.Text = instance.Country;
            txtEmail.Text = instance.Email;
            txtFax.Text = instance.Fax;
            txtPhone.Text = instance.Phone;
            txtPostalCode.Text = instance.PostalCode;
            txtWebSite.Text = instance.WebSite;
            cboRegion.Text = instance.Region;
        }
    }
    if (action == "update")
    {
        this.Title = "修改客户信息";
        btnUpdate.Visible = true;
    }
    else
    {
        this.Title = "删除客户信息";
        btnDelete.Visible = true;
    }
}
}
```

在这段代码中，首先获得名为 `action` 的页面地址参数，若 `action` 参数值等于 `insert`，则该页面用于新增记录，此时设置文档标题，然后查询数据库获得新的客户基本信息编号值，并显示出“新增”按钮；若 `action` 参数值等于 `update` 或 `delete`，则该页面用于修改或删除已有记录，将 `customerid` 的页面参数作为记录编号，查询数据库获得数据并填充到页面各个文本框中，然后设置标题，设置“更新”按钮或“删除”按钮的可见性。

使用以下代码来处理“新增”按钮的点击事件。

```
CustomerClass instance = GetInstanceFromUI();
using (IDbConnection conn = Util.CreateConnection())
{
    conn.Open();
```



```

        if (instance.Insert(conn))
        {
            this.ClientScript.RegisterStartupScript(
                this.GetType(),
                "Insert",
                @"window.alert('新增客户'基本数据成功!');
                window.returnValue = 'InsertOK' ;
                window.close();",
                true);
        }
    }
}

```

在这段代码中，程序从用户界面中获得用户输入的数据，创建一个客户记录信息对象，然后连接数据库，调用客户信息记录对象的 `Insert` 方法插入数据，若成功则输出一段 JavaScript 脚本，该脚本设置 `window` 对象的 `returnValue` 属性值为“InsertOK”，并调用 `window` 对象的 `close` 方法。

这段代码调用了 `GetInstanceFromUI` 方法，该方法用于从用户界面中获得用户输入的数据，创建一个客户信息记录对象，定义该方法的代码如下：

```

private CustomerClass GetInstanceFromUI()
{
    CustomerClass instance = new CustomerClass();
    instance.CustomerID = Convert.ToInt32(txtCustomerID.Text);
    instance.Address = txtAddress.Text;
    instance.City = txtCity.Text;
    instance.CompanyName = txtCompanyName.Text;
    instance.ContactName = txtContactName.Text;
    instance.ContactTitle = txtContactTitle.Text;
    instance.Country = txtCountry.Text;
    instance.Email = txtEmail.Text;
    instance.Fax = txtFax.Text;
    instance.Phone = txtPhone.Text;
    instance.PostalCode = txtPostalCode.Text;
    instance.WebSite = txtWebSite.Text;
    instance.Region = cboRegion.Text;
    return instance;
}

```

使用以下代码处理“修改”按钮的点击事件。

```

CustomerClass instance = GetInstanceFromUI();
if (instance != null)
{
    using (IDbConnection conn = Util.CreateConnection())
    {
        conn.Open();
        if (instance.Update(conn))
        {
            this.ClientScript.RegisterStartupScript(
                this.GetType(),
                "Update",
                @"window.alert('修改客户基本信息数据成功!');
                window.returnValue = 'UpdateOK' ;
                window.close();",
                true);
        }
    }
}
}

```

在这段代码中，程序从用户界面中获得用户输入的数据，连接数据库，然后调用客户记录信息对象的 Update 方法更新新数据库。若更新成功则输出一段 JavaScript 脚本，设置页面的返回值为“UpdateOK”并关闭窗口。

使用以下代码处理“删除”按钮的点击事件。

```
CustomerClass instance = GetInstanceFromUI();
if (instance != null)
{
    using (IDbConnection conn = Util.CreateConnection())
    {
        conn.Open();
        if (instance.Delete(conn))
        {
            this.ClientScript.RegisterStartupScript(
                this.GetType(),
                "Delete",
                @"window.alert('删除客户基本信息数据成功!');
                window.returnValue = 'DeleteOK' ;
                window.close();"
                , true);
        }
    }
}
```

在这段代码中，程序从用户界面获得数据，创建一个客户记录信息对象，调用它的 Delete 方法删除数据。若操作成功则输出一段 JavaScript 代码，设置页面的返回值为“DeleteOK”并关闭页面。

这样就完成了对 PageEditCusotmer.aspx 页面的开发。该页面运行时的用户界面如图 11-11 所示。



图 11-11 用户界面

通过 PageCustomerList.aspx 和 PageEditCusotmer.aspx 两个页面，就能实现对客户基本信息的增、删、改、查的功能。PageCustomerList.aspx 实现查询功能，而 PageEditCustomer.aspx 实现增、删、改功能。

11.3 类型使用参考说明

本节介绍 ADO.NET 中重要类型的使用参考说明。

11.3.1 System.Data.IDbConnection 接口类型

System.Data.IDbConnection 接口类型是 ADO.NET 中最基础的类型。它定义了数据库连接对象的通用编程接口，各种系统或自定义的数据库连接对象必须实现这个接口，开发者可以通过它来统一调用各种数据库连接对象。

该接口类型常用的属性如表 11-4 所示。

表 11-4 常用属性

属 性 名	数 据 类 型	说 明
ConnectionString	string	数据库连接字符串
ConnectionTimeout	int	数据库连接超时时间，单位秒。若连接某个数据库而等待服务器的时间超过这个属性指定的时间，则认为超时，系统报错
Database	string	只读属性，用于返回数据库名称
State	System.Data.ConnectionState	数据库连接的当前状态。属性值为枚举类型，其常用的可能状态有： Closed 数据库连接处于关闭状态 Open 数据库连接处于打开状态

该接口类型常用的方法如表 11-5 所示。

表 11-5 常用方法

方 法 签 名	说 明
IDbTransaction BeginTransaction() IDbTransaction BeginTransaction (IsolationLevel)	开始数据库事务处理
void Close()	关闭数据库连接，并设置数据库状态属性 State 的值为 Closed
IDbCommand CreateCommand()	创建一个数据库命令对象
void Open()	根据当前设置，连接数据库
void Dispose()	销毁对象，释放资源

11.3.2 System.Data.IDbCommand 接口类型

IDbCommand 接口表示一个数据库命令对象，通过它来执行查询和更新数据库的操作。其常用的属性如表 11-6 所示。

表 11-6 常用属性

属 性 名	数 据 类 型	说 明
CommandText	string	查询或更新数据库使用的 SQL 命令文本
CommandTimeout	int	执行命令时等待的超时时间。单位秒，默认为 30 秒
CommandType	System.Data.CommandType 枚举类型	说明 CommandType 属性值的类型，为枚举类型，可选值有： Text: SQL 文本命令，这是默认值 StoredProcedure: 存储过程名称 TableDirect: 数据表的名称
Connection	System.Data.IDbConnection	本对象使用的数据库连接对象
Parameters	System.Data.IDataParameterCollection	获得命令参数对象列表的只读属性。这个列表的成员是 System.Data.IDataParameter 类型
Transaction	System.Data.IDbTransaction	获得对象使用的事务信息对象

该类型常用的方法如表 11-7 所示。

表 11-7 常用方法

方 法 签 名	说 明
void Cancel()	试图取消当前执行中的操作
IDbDataParameter CreateParameter()	创建一个命令参数对象
int ExecuteNonQuery()	执行 SQL 命令，返回操作所影响的数据库记录行数
IDataReader ExecuteReader()	执行 SQL 查询，返回能读取查询结果的数据读取器
IDataReader ExecuteReader(CommandBehavior behavior)	执行 SQL 查询，返回数据读取器。方法的参数为 System.Data.CommandBehavior 枚举类型
Object ExecuteScalar()	查询数据库，并返回查询结果的第一行的第一列数据，忽略其他数据
void Dispose()	销毁对象，释放资源

11.3.3 System.Data.IDataReader 接口类型

IDataReader 是一种只读的只能向前的数据读取器，能快速地读取 IDbCommand 对象查询数据库所得的结果。其常用的属性如表 11-8 所示。

表 11-8 常用属性

属 性 名	数据类型	说 明
FieldCount	Int	返回查询结果的字段个数的只读属性
IsClosed	Bool	返回对象是否已经关闭了的只读属性
this[int]	object	获得当前记录的指定序号的字段值
this[string]	object	获得当前记录的指定名称的字段值

IDataReader 类型常用的方法如表 11-9 所示。

表 11-9 常用方法

方 法 签 名	说 明
void Close()	关闭对象
void Dispose()	销毁对象，释放资源
bool GetBoolean(int)	获得当前记录的指定字段的布尔类型的值
byte GetByte(int)	获得当前记录的指定字段的字节数值
char GetChar(int)	获得当前记录的指定字段的字符数值
long GetChars(int i,long fieldoffset,char[] buffer,int bufferoffset,int length)	获得当前记录的指定字段的字符数据。各个参数说明如下： i: 从 0 开始的字段列的序号 fieldoffset: 在字段值中的索引序号，从该位置开始读取数据 buffer: 保存读出数据的缓冲区 bufferoffset: 缓存区中开始保存数据的序号 length: 最大读取的字符个数 函数返回值：实际读取的字符数
string GetDataTypeName(int)	获得指定字段的数据类型名称
DateTime GetDateTime(int)	获得当前记录指定字段的日期数值
decimal GetDecimal(int)	获得当前记录指定字段的数值
double GetDouble(int)	获得当前记录指定字段的双精度浮点数
Type GetFieldType(int)	获得指定字段的数据类型
float GetFloat(int)	获得当前记录指定字段的单精度浮点数
Guid GetGuid(int)	获得当前记录指定字段的 GUID 值
short GetInt16(int)	获得当前记录指定字段的短整数值
int GetInt32(int)	获得当前记录指定字段的整数值
long GetInt64(int)	获得当前记录指定字段的长整数值
string GetName(int)	获得指定字段的名称
int GetOrdinal(string)	查找指定名称的字段编号，若未找到则抛出异常
string GetString(int)	获得当前记录指定字段的文本值
object GetValue(int)	获得当前记录指定字段的数值
int GetValues(object[] values)	用当前记录值填充对象数组，并返回读取数据的个数
bool IsDBNull(int)	判断当前记录的指定字段值是否为空
bool NextResult()	当读取批处理的 SQL 查询结果时，使数据读取器切换到下一个记录集
bool Read()	使数据读取器移动到下一条记录。如果操作成功则返回 true，若操作失败则表示已经到头了，没有下一条记录，并返回 false

11.3.4 System.Data.IDataParameter 接口类型

IDataParameter 接口表示执行数据库命令时使用的参数对象，是 IDbCommand 对象的 Parameters 属性列表的成员类型。其常用的属性如表 11-10 所示。

表 11-10 常用属性

属 性 名	数 据 类 型	说 明
DbType	System.Data.DbType	参数的数据类型
Direction	System.Data.ParameterDirection	参数的方向，表示可以只输入、只输出，或者可以双向的参数
IsNullable	bool	指明参数能否接受空值
ParameterName	string	参数名称
SourceColumn	string	参数映射的字段名
Value	object	参数值

IDataParameter 接口类型用于表示一个参数，并不实现什么其他功能，因此只有属性，没有常用方法。

11.3.5 System.Data.DataTable 类型

DataTable 类型表示在内存中的一个数据表。它包含若干个数据列和若干个数据行对象，实现了一种二维表格的数据结构。其常用的属性如表 11-11 所示。

表 11-11 常用属性

属 性 名	数 据 类 型	说 明
Columns	DataColumnCollection	属于该表的数据列对象列表，该列表的成员类型是 DataColumn
DataSet	DataSet	对象所属的 DataSet 对象
Rows	DataRowCollection	对象所含的表格行对象列表，该列表的成员类型是 DataRow 类型
TableName	string	数据表的名称

DataTable 类型常用的方法如表 11-12 所示。

表 11-12 常用方法

方 法 签 名	说 明
void AcceptChanges()	提交对数据表进行的所有更改，将更改保存到数据库中
void Clear()	清除所有的数据，删除所有的数据行
DataTable Clone()	复制对象，新的数据表具有相同的架构和约束，但没有数据，没有数据行
DataTable Copy()	复制对象，包括结构和数据
DataTableReader CreateDataReader()	在数据表的基础上创建一个数据读取器
void Dispose()	销毁对象，释放资源
void Load(IDataReader)	根据一个数据读取器加载架构和数据，方法参数就是数据读取器
DataRow NewRow()	创建一个数据行对象 DataRow 对象不能直接实例化，必须调用 DataTable 的 NewRow 方法来创建，创建后 DataRow 不属于 DataTable，还必须调用 DataTable 的 Rows 属性的 Add 方法并添加到 DataTable 中。一个 DataTable 对象创建的 DataRow 对象只能添加到该 DataTable 中，而不能添加到其他 DataTable 中

续表

方法 签 名	说 明
XmlReadMode ReadXml(Stream)	从指定的流中读取 XML 数据并填充数据表的架构和数据
XmlReadMode ReadXml(string)	从指定的文件中读取 XML 数据并填充数据表的架构和数据
XmlReadMode ReadXml(TextReader)	从指定的文本读取器中读取 XML 数据并填充数据表的架构和数据
XmlReadMode ReadXml(XmlReader)	从指定的 XML 读取器中读取 XML 数据并填充数据表的架构和数据
void ReadXmlSchema(Stream)	从指定的流中读取 XML 数据并填充数据表的架构，但不读取其中的数据
void ReadXmlSchema(string)	从指定的文件中读取 XML 数据并填充数据表的架构
void ReadXmlSchema(TextReader)	从指定文本读取器中读取 XML 数据并填充数据表的架构
void ReadXmlShcema(XmlReader)	从指定的 XML 文本读取器中读取 XML 数据并填充数据表的架构
void RejectChanges()	撤销自对象加载或最后一次调用 AcceptChanges 以来对数据表的更改
DataRow[] Select()	获得所有的表格行对象
DataRow[] Select(string)	获得与筛选条件相匹配的所有表格行对象，方法参数为筛选条件字符串，其语法类似于 SQL 语句中的 Where 子语句的内容。例如“部门=‘研发部’ And 工龄>3”
DataRow[] Select(string filterExpression , string sort)	获得与筛选条件匹配的所有表格行对象，并以指定的方式进行排序。第一个参数为筛选条件；第二个参数为排序方式，类似 SQL 语句中的 Order By 子语句的内容，例如“工龄 Desc”
void WriteXml(Stream) void WriteXml(string) void WirteXml(TextWriter) void WriteXml(XmlWriter)	将数据表的架构和数据以 XML 格式写入指定的目标。输出目标可以为流、本地文件、文本书写器或 XML 书写器
void WriteXml(Stream , XmlWriteMode) void WriteXml(string , XmlWriteMode) void WriteXml(TextWriter , XmlWriteMode) void WriteXml(XmlWriter , XmlWriteMode)	将数据表的内容以 XML 格式写入指定的目标。第一个参数为输出目标；第二个参数为输出格式，为枚举类型，可选值有： WriteSchema：只输出数据表的架构，不输出架构 IgnoreSchema：只输出数据，不输出架构 DiffGram：完整地输出
void WriteXmlSchema(Stream) void WriteXmlSchema(string) void WriteXmlSchema(TextWriter) void WriteXmlSchema(XmlWriter)	将数据表的架构以 XML 格式输出

11.3.6 System.Data.DataColumn 类型

DataRow 对象表示 DataTable 中一个数据行的架构信息，它可被独立地创建并可添加到 DataTable 对象的 Columns 列表中。该类型常用的属性如表 11-13 所示。

表 11-13 常用属性

属 性 名	数 据 类 型	说 明
AllowDBNull	bool	允许该列数据为空，默认为 true
AutoIncrement	bool	指示该列的字段值是自动递增的，默认为 false
AutoIncrementSeed	long	字段值是自动递增时的字段初始值
AutoIncrementStep	long	字段值是自动递增时的增量
Caption	string	列的标题，例如字段的中文名
ColumnName	string	列的名称，例如字段的英文名
DataType	Type	列字段值的数据类型
DateTimeMode	DataSetDateTime	数据列的日期格式数据的存储格式，为枚举类型，可选值有 Local、Unspecified、UnspecifiedLocal、Utc
DefaultValue	object	该列字段默认值
Expression	string	字段表达式
ExtendedProperties	PropertyCollection	用户自定义的属性集合
MaxLength	int	文本列字符最大个数；默认为-1，表示无限制
Ordinal	int	对象在表格列集合中的序号
ReadOnly	bool	指定数据表中该列的值是否允许修改
Table	DataTable	对象所属的数据表对象
Unique	bool	指示该字段的值是否有唯一性，不能出现重复

11.3.7 System.Data.DataRow 类型

DataRow 对象表示 DataTable 中的一行数据。其常用的属性如表 11-14 所示。

表 11-14 常用属性

属 性 名	数 据 类 型	说 明
this[DataColumn]	object	获得数据行中指定栏目的值
this[int]	object	获得指定序号的栏目的字段值
this[string]	object	获得指定名称的栏目的字段值
ItemArray	Object[]	获得或设置该数据行中所有栏目的字段值
RowState	System.Data.DataRowState	获得数据行的当前状态，为枚举类型，可选值有： Detached：该行已被创建，但还不属于任何数据表 Unchanged：该行内容没有改变 Added：该行是新加入表格行的 Deleted：该行已经被删除了 Modified：该行内容已经被修改
Table	DataTable	该行拥有其架构的 DataTable 对象

开发第一个 JavaScript 应用程序

JavaScript 是运行在 Web 浏览器中的脚本语言，是所有 Web 开发中基础技术之一，可以说没有 JavaScript 就没有现代的 Web 开发。因此 JavaScript 是任何一个 Web 开发人员的基本功。本章配套演示的程序为“第一个 JavaScript 应用程序”。

12.1 JavaScript 基本概念

JavaScript 是一种基于对象和事件驱动的客户端脚本语言，采用解释执行模式，可运行在多种软件中。一般运行在 Web 浏览器中，能在客户端操作 HTML 文档以实现动态效果，是 DHTML 的技术基础。

图 12-1 是近期全球计算机编程语言使用率排行榜变化图。

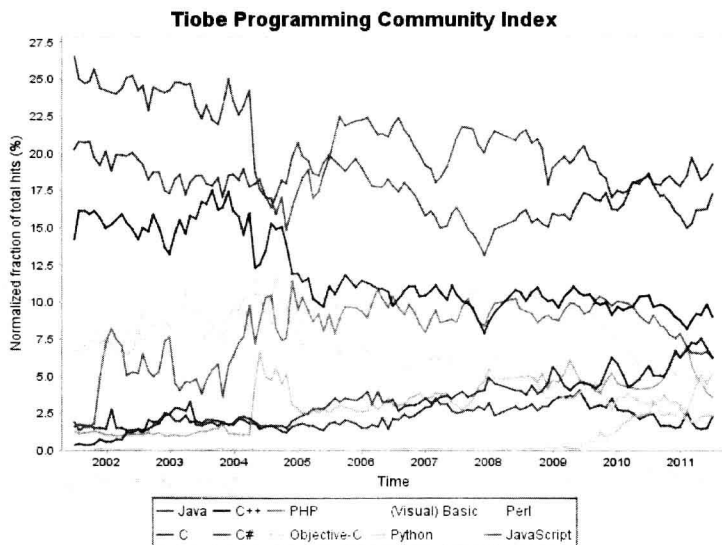


图 12-1 全球计算机编程语言使用率排行榜变化图

在 2011 年 7 月，JavaScript 使用率排在第 10 名，为 2.242%。从统计数据可以看出，JavaScript 作为一种脚本语言在业界占有一定的位置，是一种使用比较普遍的编程语言。

JavaScript 与 Java 没有什么关系，只是 JavaScript 采用了 Java 语法的子集，其他如运行环境、应用领域等完全不一样；相比而言，VBScript 与 VB 的关系很密切。

图 12-2 是 JavaScript 的运行架构图。

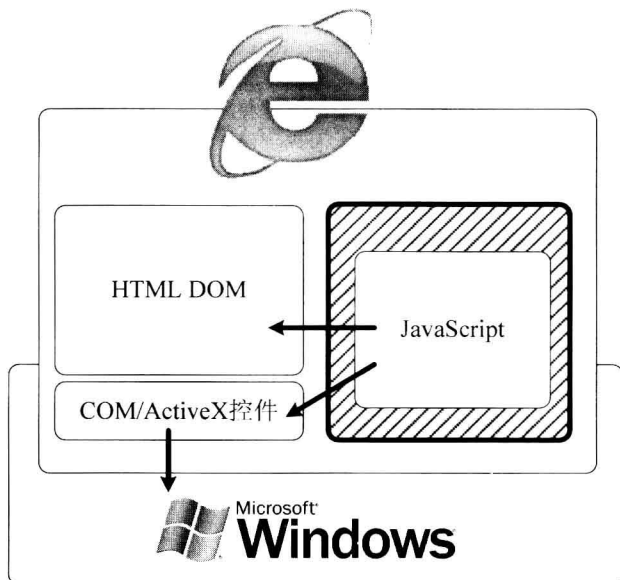


图 12-2 JavaScript 运行架构图

在该图中，JavaScript 运行在 Web 浏览器中，这里的 Web 浏览器不仅仅是 IE 浏览器，还可以是 Firefox、谷歌浏览器甚至是运行在智能手机中的浏览器。由于 JavaScript 已经是国际标准，因此 JavaScript 可以做到一次编写到处执行。

Web 浏览器内部构造了一个 JavaScript 解释执行引擎，执行引擎加载 JavaScript 脚本代码并执行它。JavaScript 运行在一个容器中，这个容器周围有厚厚的防火墙，能抵御容器中运行着的代码对 Web 浏览器中的任何非法攻击。

浏览器还维护一个 HTML DOM 并为 JavaScript 开放访问通道。JavaScript 能访问 HTML DOM 来间接地改变浏览器中当前显示的 Web 文档的内容，表现为一种动态效果，实现 DHTML 技术。

额外地，微软的 IE 浏览器还为 JavaScript 开放了访问 COM 组件/ActiveX 控件的通道，使得 JavaScript 能通过 COM/ActiveX 来增强其功能。比如 JavaScript 本身不能直接访问操作系统，但可以通过 COM/ActiveX 来间接地访问操作系统，有能力读取和修改重要的系统数据。因此这个通道成为 IE 浏览器最大的安全漏洞，互联网上很多安全事故都是由于 IE 浏览器这个安全漏洞而造成的，为此微软公司备受指责。

12.2 HTML DOM

HTML DOM 是 DOM（文档对象模型）这种软件设计模式在 HTML 中的具体实现。它用一个 HTML DOM 对象来映射 HTML 文档中的内容，是应用程序和 HTML 文档之间的代理。应用程序可以读取这些 HTML DOM 对象的属性来读取 HTML 文档的内容，也可以通过设置 HTML DOM 对象的属性或调用其成员方法来修改 HTML 文档的内容。这里的应用程序不仅仅是 JavaScript，也可以是 VB、C++、Delphi 等支持 Win32 COM 的编程语言。

以下就是一个 HTML 文档的代码。

```
<html>
  <head>
    <title>
      这是一个 HTML 文档
    </title>
  </head>
  <body>
    <h1>
      这是一个 HTML 文档
    </h1>
    <p>
      这是一个超链接
      <a
        id="link1"
        href="http://www.sohu.com"
      >
        sohu 首页
      </a>
    </p>
  </body>
</html>
```

该 HTML 文档内容的显示如图 12-3 所示。

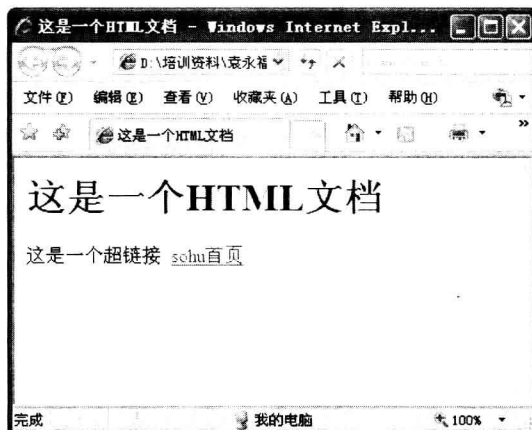


图 12-3 显示 HTML 文档内容

另外，在该 HTML 文档中编写如下的 JavaScript 代码。

```
var link =
    document.getElementById("link1");

var url = link.href;

link.innerHTML = "百度首页";

link.href="http://www.baidu.com";
```

HTML DOM 的原理图如图 12-4 所示。

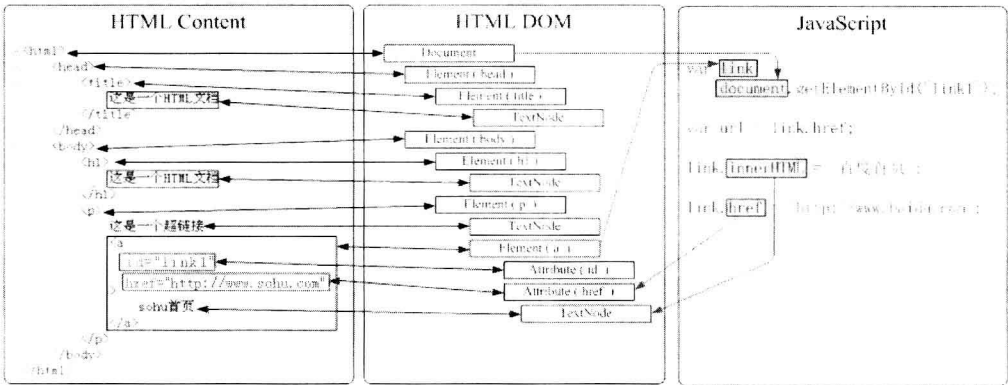


图 12-4 HTML DOM 原理图

Web 浏览器加载 HTML 文档后，解析其中的内容，并生成一个 HTML DOM 树状结构，在这个 DOM 树中，HTML 文档中的所有内容都有相对应的 HTML 文档对象。比如 Document 对象是根节点，代表整个 HTML 文档；“Element(Title)”对象对应于 HTML 文档内容“<title>这是一个 HTML 文档</title>”；“Element(a)”对象对应于“sohu 首页”；而“Attribute(href)”对象对应于“href=“http://www.sohu.com””。

HTML DOM 是可以被其他应用程序访问的，比如 JavaScript 就可以访问。在这段 JavaScript 代码中，“document”全局对象能直接访问 HTML DOM 中的 Document 对象；通过调用它的“getElementById”方法能立即定位“Element(a)”对象并设置到变量“link”中。

在 JavaScript 代码“link.innerHTML = “百度首页;””中，代码设置“link”变量指向的“Element (a)”元素的 innerHTML 属性值，修改了 HTML DOM；而 Web 浏览器能立刻感知到 HTML DOM 被修改了，然后更新当前显示的 HTML 文档的内容。这样将文档中的“这是一个超链接 [sohu 首页](http://www.sohu.com)”迅速更新为“这是一个超链接 [百度首页](http://www.sohu.com)”，也就是相当于将 HTML 文档片段“sohu 首 页”修改为“百度首页”。

类似地，JavaScript 代码“link.href=“http://www.baidu.com;””也修改了 HTML DOM 中的内容，Web 浏览器会实时地更新当前显示的 HTML 文档，相当于将 HTML 文档片段“<a

`id="link1" href="http://www.sohu.com" >` 修改为 ``，也就是设置了这个超链接的目标地址，虽然不会改变 HTML 文档的用户界面，但会修改其行为。当用户单击这个超链接时，新打开的网页地址不再是“http://www.sohu.com”而是“http://www.baidu.com”。

从这个范例看，HTML DOM 是介于 HTML 文档和 JavaScript 之间的，是 HTML 文档在 JavaScript 前面的代理。使用 HTML DOM，JavaScript 就能任意修改 HTML 文档的内容，在用户界面上呈现出一种动态效果，实现 DHTML 技术。

12.3 JavaScript 语法

JavaScript 只参考了类似 Java 的一部分语法，但它是一种独立的编程语言，它是基于对象但不是面向对象的，不支持对象的封装、继承和重载，因此不太复杂。下面介绍 JavaScript 的基本语法。

12.3.1 数据类型

JavaScript 支持以下数据类型。

(1) 字符串

表示一段文本，在代码中使用两个匹配的双引号或单引号进行定义，例如“这是一段文本”，或者“'这是一段文本'”，一般都采用双引号进行定义。JavaScript 没有字符数据类型。

(2) 布尔值

表示一个非真即假、两种状态的数值。它的值只能等于 `true` 或 `false`。

(3) 数值

表示一个数字型的数值。在 JavaScript 中不区分整数、浮点数等类型，统一当做数值数据类型。在代码中可以直接使用十进制数来定义数值数据，也可以使用“0x16 进制字符”的样式来定义数值数据，比如定义“0xf”等于十进制的“15”。

(4) 数组

表示由若干个相同的数据类型组成的数组。以下代码定义了两个数组。

```
var names = { "张三" , "李四" , "王五" };
var args = { 9 , 23 , 42 };
```

(5) 对象

表示引用了一个编程对象。

(6) null 类型

在 JavaScript 中，`null` 是一种数据类型，它只有一个名为“`null`”的值，可以对它使用 `typeof`

运算符，例如 “typeof(null)” 返回的是 object。

(7) undefine 类型

当某个变量指向了对象的不存在的属性或声明了变量但从未赋值时，该变量就是 undefine 类型，可以使用 typeof 进行类型判断，也就是 “typeof(变量)== undefine”。

JavaScript 是一种弱类型的语言，在定义变量时是无法指定变量类型的。以下代码定义了一个变量。

```
var name = "张三";
```

在这段代码中，关键字 “var” 说明正在定义一个变量；“name” 定义了变量名称；“=”张三” 初始化了变量值。

由于变量没有数据类型，因此可以在运行时向变量设置各种各样的数据，例如以下代码是可以运行的。

```
var name = "张三" ;  
name = 2.343 ;  
name = true ;  
name = 0xf9 ;
```

虽然 JavaScript 提供这种功能，但为了代码的可读性，尽量避免这种一个变量保存多种不同意义的值的情况，而且变量名应该和其功能相关。

12.3.2 运算符

JavaScript 支持不少运算符，主要有以下四种。

1. 算术运算符

算术运算符如表 12-1 所示。

表 12-1 算术运算符

符 号	描 述
-	负值，对数值取负
++	递增，让数值变量自动增 1。例如 var age = 20 ; age ++ ; 执行后 age 的值等于 21
--	递减，让数值变量自动减 1。例如 var age = 20 ; age -- ; 执行代码后 age 的值等于 19
*	乘法
/	除法
%	取模运算

续表

符 号	描 述
+	加法
-	减法

2. 逻辑运算符

逻辑运算符如表 12-2 所示。

表 12-2 逻辑运算符

符 号	描 述
!	逻辑非，例如 <code>var state = true ;</code> <code>state = ! state ;</code> 代码执行后，state 的值为 false
<	小于
>	大于
<=	小于等于
>=	大于等于
==	等于，在进行数值比较时会进行一些默认的数据类型转换
!=	不等于
&&	逻辑与
	逻辑或
?:	条件（三元运算符）
,	逗号
===	严格相等，且两者的数据类型也必须一样
!==	非严格相等

3. 位运算符

位运算符如表 12-3 所示。

表 12-3 位运算符

符 号	描 述
~	按位取反
<<	按位左移，相当于乘以 2
>>	按位右移，相当于除以 2
>>>	无符号右移
&	按位与
^	按位异或
	按位或

4. 杂项运算符

杂项运算符如表 12-4 所示。

表 12-4 杂项运算符

符 号	描 述
delete	删除
typeof	获得变量的数据类型的名称。只可能返回字符串"number"、"string"、"boolean"、"object"、"function" 和 "undefined"。例如 var v = "张三"; var tn = typeof(v); 此时 tn 的值为 "string"
void	执行表达式但不返回表达式的值，统一返回 undefined。例如 var v = 4 > 5; 此时 v 的值为 false var v = void 4 > 5; 此时 v 的值为 undefined
instanceof	instanceof
new	new
in	in

12.3.3 条件判断语法结构

JavaScript 支持多种条件判断语法结构，其中包括 if 语句和 switch 语句。

1. if 语句

JavaScript 支持 if 条件判断语句，其用法为：

```
if ( 判断条件 )
{
    执行的代码;
}
else
{
    执行的代码;
}
```

也支持 “if else if” 多重条件判断语句。

2. switch 语句

JavaScript 支持 switch 条件判断语句，其用法为：

```
switch ( 常量、变量或表达式 )
{
case 常量 1:
    执行的代码;
    break;
```



```

case 常量 2:
    执行的代码;
    break;
default:
    执行的代码;
}

```

与 C# 不一样，在 switch 语句中的 case 子语句中可以不出现 break 关键字。若程序执行中遇到 break 关键字则立即跳出 switch 语句，否则继续执行后续的 case 子语句中的代码。

12.3.4 循环语法结构

JavaScript 支持多种循环语法结构，其中包括 for 语句、for-in 语句、while 语句、do-while 语句、break 语句和 continue 语句。

1. for 语句

JavaScript 支持 for 循环语句结构，其用法为：

```

for ( 初始化代码 ; 循环继续执行判断条件 ; 一次循环后执行的代码 )
{
    循环中执行的代码;
}

```

以下代码就演示了使用 for 语句遍历数组。

```

var names = {"张三","李四","王五","赵六"};
for ( index = 0 ; index < 4; index ++ )
{
    document.write ( names[ index ] ) ;
}

```

2. for-in 语句

JavaScript 支持独特的 for-in 语句结构，其功能类似 C# 中的 foreach 语句。其用法为：

```

for ( 变量 in 循环主体对象可以为普通对象或数组 )
{
    循环中执行的代码;
}

```

若循环主体对象为普通对象，则该循环遍历该主体对象的所有成员属性值，而且无法事先确定其顺序；若循环主体对象为数组，则遍历所有的数组元素值，是按顺序遍历的。

以下代码就演示了使用 for-in 语句遍历数组。

```

var names = {"张三","李四","王五","赵六"};
for ( name in names )
{
    document.write ( name ) ;
}

```

3. while 语句

JavaScript 支持 while 语句，其用法为：

```
while ( 执行循环条件判断表达式 )
{
    循环中执行的代码 ;
}
```

在 while 语句中，每次循环前都执行循环条件判断表达式，若通过则进行依次循环，否则退出循环。

4. do-while 语句

JavaScript 支持 do-while 语句，其用法为：

```
do
{
}while ( 循环条件判断表达式 ) ;
```

当执行 do-while 中的第一次循环时是不会执行循环条件判断表达式的，此后每次执行循环前都会执行判断表达式，注意 while 语句后面有分号。

5. break 语句

在 JavaScript 中 break 语句具有两种作用。

(1) 在所有的循环体执行的语句中都可以加上 break 语句，当程序执行到 break 语句时会无条件地立即退出循环体。

(2) 在 switch 语法结构中用于立即退出 switch 语法结构。

6. continue 语句

在所有的循环体执行的语句中都可以加上 continue 语句，当程序执行到 continue 语句时都会无条件地跳过循环体中的其他语句，但仍然会执行必要的循环条件判断语句，然后接着进行下一次循环。

12.3.5 异常处理语法结构

JavaScript 支持异常处理，能抛出异常和接受异常。

1. throw 语句

throw 语句能抛出异常，其用法为：

```
throw 表达式 ;
```

以下代码就能抛出异常。

```
if ( age < 0 )
    throw "年龄不能小于 0" ;
else if ( age > 150 )
    throw "年龄不能大于 150" ;
```

当变量 age 的值为 1000 时，在 IE 浏览器中运行这段 JavaScript 会报错，并显示如图 12-5 所

示的消息框。

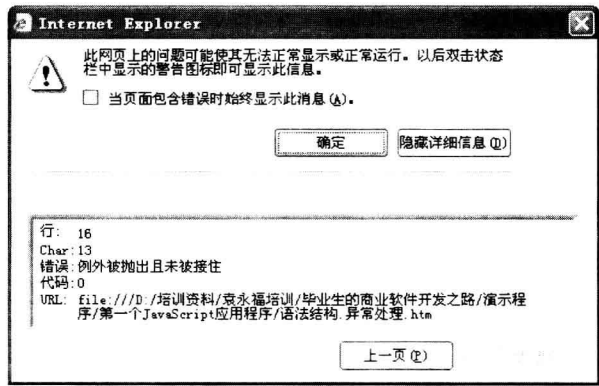


图 12-5 消息框

2. try-catch-finally 结构语句

try-catch-finally 语句能捕获异常并进行处理，其用法为：

```
try
{
    程序功能性代码 ;
}
catch ( 异常数据变量名 )
{
    处理异常的代码 ;
}
finally
{
    无论有没有发生异常都会执行的代码；
}
```

其中异常数据变量的初始值就是 throw 语句抛出的数值，可以是字符串、数字或其他数据。finally 语句块是可选的，可以没有该语句块。

以下代码就能处理异常。

```
var result = "";
try
{
    var age = 1000 ;
    if ( age < 0 )
        throw "年龄不能小于 0!!!";
    else if ( age > 150 )
        throw "年龄不能大于 150!!!";
    result = result + "\r\n接着执行";
}
catch ( exception )
{
    //此时 exception 变量值就是 throw 语句中设置的值
    result = result + "\r\n" + exception;
}
```

```
finally
{
    result = result + "\r\n结束" ;
}
alert(result);
```

在这段代码中，由于 age 设置为 1000 大于 150，于是调用了语句 “throw "年龄不能大于 150";”，则程序立即跳转到 catch 块，并自动设置变量 exception 的值为 “年龄不能大于 150”，这样代码 “result = result + "\r\n 接着执行”;” 就被忽略掉了。

无论有没有发生异常，finally 结构中的语句都会执行，因此此处最后执行的代码是 “result = result + "\r\n 结束”;”。

在 IE 浏览器中若执行这段代码，则会显示如图 12-6 所示的消息框。



图 12-6 消息框

12.3.6 其他语法结构

1. with 语句

with 语句用于为其他语句设置默认对象，其用法为：

```
with ( 变量名 )
{
    其他语句，其中上面的变量名指定的对象就是这段语句中使用的默认对象 ;
}
```

例如，JavaScript 中存在全局对象 Math，其中包含了数值运算的方法，以下代码就能调用它的成员。

```
x = Math.cos(3 * Math.PI) + Math.sin(Math.LN10)
y = Math.tan(14 * Math.E)
```

而使用 with 语句，能使代码变得短小精悍，其范例如下：

```
with (Math)
{
    x = cos(3 * PI) + sin (LN10)
    y = tan(14 * E)
}
```

2. function 语句

function 语句用于定义一个函数，其用法如下：

```
function 函数名( 参数 1 , 参数 2 , 参数 3 )
{
```

```
    函数功能代码 ;
```

```
}
```

以下代码定义了一个函数。

```
function sum ( num1 , num2 )
{
    return num1 + num2 ;
}
```

在 JavaScript 中调用函数时，必须在函数后面添加圆括号和所需的参数，不带圆括号而调用函数将返回所定义该函数的 JavaScript 脚本代码文本，而不是其内部代码的执行结果。

例如，对于以下代码：

```
function sum(num1, num2)
{
    return num1 + num2;
}

function 正常执行 sum 函数()
{
    var result = sum(2, 3);
    alert(result);
}

function 不带圆括号地执行 sum 函数()
{
    var result = sum;
    alert(result);
}
```

若运行“正常执行 sum 函数”，则系统会显示如图 12-7 所示的消息框。若运行“不带圆括号地执行 sum 函数”，则系统会显示如图 12-8 所示的消息框。

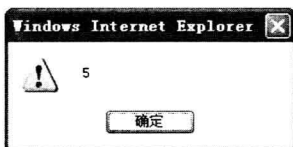


图 12-7 消息框

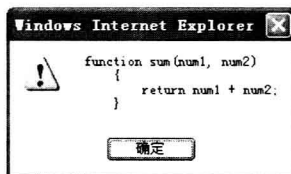


图 12-8 消息框

3. return 语句

return 语句用于立即退出一个函数。

12.4 系统预定义对象

JavaScript 是一个基于对象的编程语言，能使用一些对象进行编程，这些对象分为系统实例对象类型，系统全局对象和用户自定义对象。

12.4.1 系统实例对象

JavaScript 系统虽然已经预定义了一些实例对象，但必须创建一个对象实例才能使用这些对象。下面介绍常用的对象。

1. string 对象

string 对象表示一个字符串，它提供 length 属性用于返回字符串的长度，还提供很多成员方法，常用的如表 12-5 所示。

表 12-5 string 对象提供的常用成员方法

成员方法	功能
charAt (index)	获得字符串中指定位置处的字符。参数 index 是整数，从 0 开始计算序号
charCodeAt (index)	获得字符串中指定位置处的字符的 Unicode 编码
concat (字符串 1 , 字符串 2 , ...)	用于连接两个或多个字符串，并返回新的字符串，这个成员方法的参数个数不固定，可以为一个或多个。例如 var str = "12345"; var str2 = str.concat("789"); 执行后，变量 str2 的值为"12345789"，注意原先的变量 str 的内容没有改变
indexOf (字符串 , startIndex)	用于返回字符串中第一次出现子字符串的位置，第一个参数为要查找的子字符串，第二个参数是可选的，指定查找的起始位置，若未指定则从头查找。例如 var str = "1234567890"; var index = str.indexOf("789"); 执行后变量 index 的值为 6
lastIndexOf(子字符串, startIndex)	从字符串结尾开始向前找，查找子字符串最后出现的位置。第一个参数为要查找的子字符串；第二个参数是可选的，就是开始查找的位置，若省略则从字符串的末尾开始查找
match (正则表达式)	使用正则表达式进行字符串查找，若没有找到则返回 null，若找到则返回一个数组，并更新全局 RegExp 对象的属性以反映匹配结果
replace(旧字符串/正则表达式, 新字符串)	进行字符串替换，并返回替换后的字符串。本函数第一个参数是一个旧字符串或正则表达式，第二个参数是要替换成的新字符串，本函数不会改变字符串本身的值，但返回值为执行替换操作后的新字符串

2. Date 对象

该对象表示一个日期和时间数据，精确到毫秒。以下代码能创建 Date 对象。

```
var dtm1 = new Date( );           // 获得当前日期和时间
var dtm2 = new Date( 1980 , 1 , 3 ); // 1980 年 1 月 3 号
var dtm3 = new Date( 1980 , 1 , 3 , 21 , 38 , 19 ); // 1980 年 1 月 3 号 21 点 38 分 19 秒
```

Date 对象常用的成员方法如表 12-6 所示。

表 12-6 Data 对象常用的成员方法

getDate()	返回这个日期在当前月中的号数，从 1 到 31
setDate(numDate)	设置日期在当前月中的号数，参数有效取值范围为 1 到 31
getDay()	返回一个介于 0 到 6 的整数，表示一周中的某天。返回 0 表示星期一，返回 1 表示星期二，以此类推，返回 6 表示星期天
getFullYear()	返回一个完整的表示年数的整数。例如，对于 1980 年返回值就是 1980，而不是 80
setFullYear(numYear)	设置日期中完整的年数，参数为一个整数，表示年份
getMonth() setMonth(numMonth)	获得或设置日期数据中的月份值，该月份值从 0 到 11
getHours() setHours(numHours)	获得或设置时间数据中的 24 小时制的小时数
getMinutes() setMinutes(numMinutes)	获得或设置时间数据中的分数
getSeconds() setSeconds(numSecionds)	获得或设置时间数据中的秒数
getTime() setTime(milliseconds)	获得或设置时刻值。时刻值是一个整数，表示从 1970 年 1 月 1 日开始所经过的毫秒数
toString()	获得一个表示日期时间数值的字符串
Date.parse(value)	这是一个静态函数，能将一个字符串解析成一个日期数据。例如，执行以下代码： <code>var dtm = Date.parse("1980-2-14 13:23:53");</code> 则能获得一个表示 1980 年 2 月 14 号 13 时 23 分 53 秒的 Date 值

3. Array 对象

该对象表示一个数组。以下代码就定义了数组对象。

```
var array1 = new Array( ); //定义了一个没有元素的数组
var array2 = new Array( 100 ); //定义了一个已经有 100 个元素的数组
var array3 = new Array("张三" , "李四","王五","赵六" ); //定义了一个数组并已经包含了几个元素
```

在 JavaScript 中，数组的下标是从 0 开始计算的。它由 length 属性返回数组的元素个数。

4. ActiveXObject 对象

该对象表示一个 ActiveX 对象引用。这是 IE 浏览器所特别支持的，而其他浏览器可能不支持这种类型。以下代码定义了 ActiveXObject 对象实例。

```
var app = new ActiveXObject("Excel.Application") ;
var app2 = new ActiveXObject("Excel.Sheet") ;
```

ActiveXObject 对象没有确定的属性和成员方法，完全依赖所创建的 ActiveX 对象的属性和方法。

5. 正则表达式对象

本对象包含正则表达式模式和一些配置标识。以下代码定义了一个正则表达式对象。

```
var re = new RegExp("Name" , "i") ;
```

或者

```
var re = /Name/i ;
```

在创建对象时，第一个参数就是正则表达式模式，第二个参数是行为选项。目前支持的行为选项有“g”：全文查找出现的所有模式，“i”：忽略大小写，“m”：多行查找。

正则表达式对象和全局对象 `RegExp` 是不同的，正则表达式只是定义一些模式匹配的信息，没有其他实际功能，能当做字符串对象的 `match` 方法的参数值；而全局对象 `RegExp` 的属性包含执行正则表达式后的匹配结果。

例如，以下代码能执行正则表达式。

```
var txt = "name Name naaame" ;  
var re = new RegExp("Name" , "i");  
var result = txt.match( re ) ;
```

此时 `result` 变量就是匹配结果。

6. function 对象

该对象表示一个方法。以下代码定义了一个 `function` 对象。

```
var mul = function (num1, num2)  
{  
    return num1 * num2;  
}
```

定义函数后，其他地方仍然按照正常方式调用，比如对于上述代码可以执行“`var result = mul(2,3)`”。

7. arguments 对象

该对象表示正在执行的函数的参数，这种类型不需要进行实例化，在函数体中默认已经存在。它提供 `length` 属性来获得参数个数；提供“`arguments[index]`”的写法来获得指定序号处的参数，序号是从 0 开始计算的。

12.4.2 系统全局对象

JavaScript 中系统预定义了一些全局对象，这些全局对象可以在代码中直接使用。下面介绍常用的全局对象。

1. Global 对象

这是一个系统全局对象，在代码中可以使用“`Global.成员名称`”调用，也可以直呼其名地来

调用，比如可以使用 “Global.parseFloat("123.45")”，也可以使用 “parseFloat("123.45”)”。

该全局对象提供的属性如表 12-7 所示。

表 12-7 Global 对象提供的属性

属 性	功 能
Infinity	返回一个数值，表示无穷大
NaN	返回一个非数字的特殊值

Global 对象提供的方法如表 12-8 所示。

表 12-8 Global 对象提供的方法

方 法	功 能
escape (charString)	对字符串进行编码，使其能在所有的计算机上可读。该函数能对文本中所有的空格、标点、重音符号以及其他非 ASCII 字符都用 “%xx” 十六进制编码代替，例如对于空格返回 “%20”，对于字符值大于 255 的以 “%uxxxx” 来代替 escape 方法不能用来对统一资源标示码（URI）进行编码。对其编码应使用 encodeURIComponent 和 encodeURIComponent 方法
unescape (charString)	解码用 escape 函数编码的字符串。本函数是 escape 的反函数
eval (codeString)	执行 JavaScript 代码。例如可以执行以下代码： eval("var dtm = new Date()); 执行这行代码后，后面的 JavaScript 代码就能访问动态创建的 “dtm” 变量值
isNaN (numValue)	用于检查指定的数值是否是特殊值 NaN。若等于特殊值 NaN 则返回 true，否则返回 false
parseFloat (numString)	将一个字符串解析成一个浮点数并返回解析所得的浮点数，若解析失败则返回 NaN，可以使用 isNaN 函数来判断解析结果。例如代码 “parseFloat("12.3")” 返回 12.3，而代码 “parseFloat("abc")” 返回 NaN
parseInt (numString)	将一个字符串解析成一个整数并返回解析结果，如解析失败则返回 NaN。若字符串前缀为 “0x” 则当做十六进制，若前缀为数字字符 “0” 则当做八进制，否则当做十进制

2. Math 对象

Math 对象提供一些用于数值运算的成员，必须通过 “Math.成员名称” 来调用它的成员。该对象提供的属性如表 12-9 所示。

表 12-9 Math 对象提供的属性

属 性	功 能
E	返回欧拉常数，自然对数的底，约等于 2.718
LN2	返回 2 的自然底数，约等于 0.693
LN10	返回 10 的自然底数，约等于 2.302
LOG2E	返回以 2 为底的 e 的对数，约等于 1.442
LOG10E	返回以 10 为底的 e 的对数，约等于 0.434

续表

属 性	功 能
PI	返回圆周率，约等于 3.14159
SQRT1_2	返回 0.5 的平方根，约等于 0.707
SQRT2	返回 2 的平方根，约等于 1.414

Math 对象提供的方法如表 12-10 所示。

表 12-10 Math 对象提供的方法

方 法	功 能
abs(x)	返回数值的绝对值
acos(x)	返回数值的反余弦值
asin(x)	返回数值的反正弦值
atan(x)	以介于-PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值
atan2(y,x)	返回从 x 轴到点 (x,y) 的角度（介于-PI/2 与 PI/2 弧度之间）
ceil(x)	对数值进行上舍入
cos(x)	返回数值的余弦
exp(x)	返回 e 的指数
floor(x)	对数值进行下舍入
log(x)	返回数值的自然对数（底为 e）
max(x,y)	返回 x 和 y 中的最高值
min(x,y)	返回 x 和 y 中的最低值
pow(x,y)	返回 x 的 y 次幂
random()	返回 0 ~ 1 之间的随机数
round(x)	把数四舍五入为最接近的整数
sin(x)	返回数值的正弦
sqrt(x)	返回数值的平方根
tan(x)	返回数值的正切

3. RegExp 对象

本全局对象保存了最后一次执行正则表达式模式匹配的执行结果。该对象提供的属性如表 12-11 所示。

表 12-11 RegExp 对象提供的属性

属 性	功 能
index	第一个匹配项在字符串中的开始位置
input	最近一次被查找的字符串
lastIndex	返回被查找字符串中下一次成功匹配的位置
lastMatch	返回最后一次匹配的字符串
lastParen	返回最后的子匹配

续表

属 性	功 能
leftContent	返回从被查找的字符串的开始位置到最后匹配的位置中的子字符串
rightContent	返回从被查找字符串的末尾到最后匹配的位置之间的子字符串
\$1 - \$9	返回 9 个在模式匹配中找到的子字符串

4. document 对象

document 对象是 HTML DOM 提供的对象，它代表整个 HTML 文档，是 JavaScript 访问 HTML DOM 的入口点。下面介绍该对象常用的成员。

(1) all[] 集合

返回文档中对所有 HTML 元素的引用。若 HTML 文档中有个 id 号为 element1 的元素，可以使用 “document.all[“element1”]” 来快速访问这个元素，对于 IE 浏览器也可以使用 “document.all.element1” 来快速访问该元素。当然也可以使用 “document.all[0]” 的方式来获得元素，但元素在文档中的序号一般是不好确定的，因此用得不多。

若 HTML 文档中存在以下内容：

```
<a id="element1" href="www.sohu.com">SHOU 首页</a>
<a href ="www.baidu.com">百度首页</a>
```

则执行以下 JavaScript 代码：

```
var element = document.all["element1"];
alert(element.innerHTML);
```

系统会显示如图 12-9 所示的消息框。



图 12-9 消息框

若在 IE 浏览器中执行以下 JavaScript 代码：

```
element = document.all.element1;
alert(element.outerHTML);
```

效果也是一样的，只是这段代码只能在 IE 浏览器中运行，在其他浏览器中可能无效。

(2) cookie 属性

用于获得浏览器本地 cookie 数据集合。

(3) title 属性

用于获得或设置当前 HTML 文档标题，在脚本代码修改了 title 属性值后，浏览器窗体标题也会立即更新。若 HTML 文档包含以下 HTML 代码：

```
<title>对象.document</title>
```

则 IE 浏览器标题栏如图 12-10 所示。



图 12-10 标题栏

若执行以下 JavaScript 代码：

```
alert("当前标题是：" + document.title);
```

则系统显示如图 12-11 所示的消息对话框。



图 12-11 消息对话框

若执行以下 JavaScript 代码：

```
document.title = "修改了标题";
```

则执行后浏览器的标题栏如图 12-12 所示。



图 12-12 标题栏

(4) URL 属性

用于获得或设置当前文档的 URL 地址。用脚本代码修改了 URL 属性值后，浏览器会从新的 URL 地址重新加载当前显示的 HTML 文档。

以下代码显示了当前文档的 URL 地址，并指向新的网站，浏览器会自动打开这个网址。

```
alert("当前 URL 是：" + document.URL);  
document.URL = "http://www.sohu.com";
```

(5) getElementById 方法

用于返回第一个指定 id 的对象。参数为一个字符串格式的 id 值。

例如，HTML 文档中有以下 HTML 片段：

```
<a id="element1" href="http://www.sohu.com">SHOU 首页</a>
```

则可以执行以下 JavaScript 代码：

```
var element = document.getElementById("element1");  
alert(element.outerHTML);
```

浏览器会弹出如图 12-13 所示的对话框。



图 12-13 询问对话框

(6) getElementByName 方法

用于返回具有指定 name 属性值的对象集合，参数为名称值。

(7) getElementByTagName 方法

用于返回具有指定标签名称的对象集合，参数为标签名称值。

(8) Write 方法

用于向文档输出 HTML 代码，参数为要输出的内容。

(9) Writeln 方法

用于向文档输出 HTML 代码，而且最后输出一个换行符，参数为要输出的内容。

5. window 对象

window 全局对象代表浏览器窗体，下面介绍其常用的成员。

(1) alert 方法

显示一个消息对话框，函数参数就是要显示的消息内容。若执行以下代码：

```
window.alert("这里显示了一个消息!");
```

则会显示如图 12-14 所示的对话框。



图 12-14 消息对话框

(2) close 方法

试图关闭窗口体，此时浏览器可能会显示如图 12-15 所示的一个确定关闭对话框，让用户确认操作。

- confirm 方法

显示一个带有“确定”、“取消”两个按钮的对话框，让用户进行选择。本方法参数为消息文本，若用户确定操作则函数返回 true，否则返回 false。

例如执行以下代码：

```
var result = window.confirm("请确认是否继续操作?");
```

则系统会弹出如图 12-16 所示的对话框。



图 12-15 确认对话框

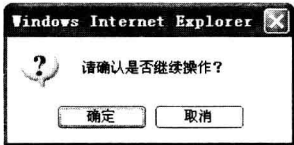


图 12-16 确认对话框

若用户单击“确认”按钮关闭对话框，则函数返回值为 `true`；若单击“取消”按钮或单击窗口右上角的“关闭”按钮则函数返回 `false`。

- `prompt` 方法

这个方法会弹出一个带有文本输入框的对话框，供用户输入一些数据。函数有两个参数，第一个参数为提示文本，第二个参数为默认值，函数返回用户输入的结果。

例如执行以下代码：

```
var result = window.prompt("请输入您的姓名:", "袁永福");
```

则会显示如图 12-17 所示的对话框。

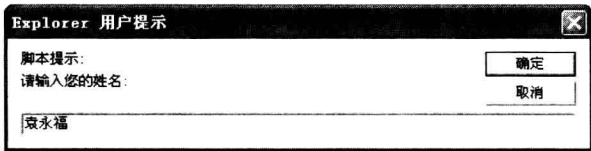


图 12-17 提示对话框

若用户输入数据，即使输入了空数据，然后单击“确定”按钮关闭对话框，则 `prompt` 函数返回用户输入的文本；若用户单击“取消”按钮关闭对话框，则 `prompt` 函数返回 `null` 值。

- `setTimeout` 方法和 `clearTimeout` 方法

这是一对作用相反的函数。

`setTimeout` 函数用于延时执行一段 JavaScript 代码。该函数有两个参数，第一个参数为 JavaScript 代码文本，第二个为整数，以毫秒为单位地延时时间长度，函数返回值是一个句柄。

`clearTimeout` 函数用于取消这种操作，函数参数为一个句柄，就是 `setTimeout` 函数的返回值。

下面的 JavaScript 代码就展示了这两个函数的使用。

```
var id_Timeout = null;
function Test_window_setTimeout() {
    id_Timeout = window.setTimeout("window.alert('延时 3 秒显示的文本')", 3000);
}
function Test_window_clearTimeout() {
    if (id_Timeout != null) {
```

```
        window.clearTimeout(id_Timeout);
    }
}
```

对于这段代码，若执行 Test_Window_setTimeout 函数，则浏览器会延时 3 秒后显示一个内容为“延时 3 秒显示的文本”的消息对话框。若在这 3 秒的等待时间中执行了 Test_window_clearTimeout 函数则会取消这个延时操作，3 秒以后不会显示消息框。

- setInterval 函数和 clearInterval 函数

这是一对作用相反的函数。

setInterval 函数用于定期执行一段 JavaScript 脚本。该函数第一个参数就是要执行的 JavaScript 脚本代码文本，第二个参数就是以毫秒为单位的时间周期，函数返回值为一个句柄。

clearInterval 函数用于取消定期执行代码，函数的参数就是 setInterval 函数的返回值。

下面的 JavaScript 代码就展示了如何使用 setInterval 函数和 clearInterval 函数。

```
var id_Interval = null;
function Test_window_setInterval() {
    id_Interval = window.setInterval("window.alert('每隔 3 秒显示的文本')",
3000);
}
function Test_window_clearInterval() {
    if (id_Interval != null) {
        window.clearInterval(id_Interval);
    }
}
```

对于这段代码，若执行 Test_Window_setInterval 函数，则浏览器会每隔 3 秒显示一个消息对话框，永不停止，直到关闭浏览器程序。若执行 Test_window_clearInterval 函数，则会停止这个定期显示消息框的操作。

- showModalDialog 函数、dialogArguments 属性和 returnValue 属性

这三个成员都是用于显示和管理网页对话框的。

showModalDialog 函数能显示一个模式网页对话框，本函数有 3 个参数。

第一个参数为网页文件 URL 地址，可以为绝对路径或相对路径。

第二个参数为对话框参数，参数类型可以为任何类型，在对话框网页内部可以使用 window 的 dialogArguments 的属性值获得对话框参数。

第三个参数为一个字符串，用于控制对话框的显示样式，采用“设置名:值;名:值”的样式，例如“dialogHeight:200px;dialogWidth:200px”。支持的设置项目如表 12-12 所示。

表 12-12 设置项目

名 称	数 值	说 明
dialogHeight	表示度量的文本，比如“200px”	对话框的窗体高度
dialogWidth	表示度量的文本	对话框的窗体宽度
dialogLeft	表示度量的文本	对话框的左端位置

续表

名 称	数 值	说 明
dialogTop	表示度量的文本	对话框的顶端位置
Center	yes 或 no	是否在屏幕中央显示对话框
Edge	sunken 或 raised	对话框窗体边框样式
resizable	yes 或 no	是否允许用户改变对话框大小
Scroll	yes 或 no	是否允许对话框显示滚动条
Status	yes 或 no	是否显示对话框中的状态栏

本函数有返回值，在对话框网页中设置 window 对象的 returnValue 属性值就能设置 showModalDialog 函数的返回值。

以下代码就调用了 showModalDialog 函数。

```
var result = window.showModalDialog(
    "人员列表对话框.htm",
    "张三,李四,王五",
    "dialogHeight:200px;dialogWidth:200px:center=yes;resizable:no;
    scroll:no");
```

12.5 JavaScript 代码文件

JavaScript 代码既可以嵌入在 HTML 文档中，被包含在 script 标签中，还可以写在一个单独的扩展名为 js 的 JavaScript 代码文件中，然后被 script 标签引用。

例如，有一段 JavaScript 代码：

```
alert("显示一个消息");
```

若嵌入在 HTML 页面中，则可以写成：

```
<script language="javascript" >
    alert("显示一个消息");
</script>
```

若使用 JavaScript 代码文件方式，则可以建立一个名为“demo.js”的纯文本文件，文件内容如下：

```
alert("显示一个消息");
```

然后在 HTML 页面中使用以下代码引用这个 JS 文件：

```
<script language="javascript" src="demo.js">
</script>
```

这两种写法所实现的功能一样。

一般来说，小段的或和页面具体功能密切的 JavaScript 代码可以嵌入在 HTML 页面中，但对于大段的，或者通用的能为其他页面所调用的 JavaScript 代码可以写在一个单端的 JS 文件中，然后为多个页面引用。

12.5.1 文本编码格式

使用 JS 文件还需要注意文本编码格式的问题。一般来说, ASPX 页面输出的 HTML 页面采用 UTF-8 编码格式, 而 JS 文件有可能是 UTF-8、GB2312 或 ANSI 编码格式。当 JS 文件的编码格式和引用它的 HTML 文档的编码格式不一致时, 很容易出现中文乱码的错误, 甚至 JS 中的中文注释也会导致莫名其妙的 JavaScript 语法错误。

而对于嵌入在 HTML 页面中的 JavaScript 代码没有这种问题, 因为两者同属一个文本文件, 编码格式肯定一样。

不同版本的 VS.NET, 其生成的 ASPX 页面和 JS 文件的编码格式是不一样的。例如, 对于 VS.NET 2005 生成的文件采用 GB2312 格式, 而对于 VS.NET 2010 生成的文件是 UTF-8 格式的。

在实践中, 很多 JS 文件是复制使用的。例如, 从一个 VS.NET 2005 的项目中复制了 JS 文件到 VS.NET 2010 项目中, 或者反过来复制使用, 此时很容易由于编码不一致而导致 JS 错误。

因此在开发中要让 HTML 页面和 JS 文件采用相同的文本编码格式。

一般来说, 在 JavaScript 中编写单行注释时需要多少行就写多少行, 例如以下代码:

```
//检查用户名
if (ValidateUserName(
    document.getElementsByName("userName")[0],
    document.getElementById('lblUserName')) == false) {
    result = false ;
}
```

对于 JS 文件为了避免由于中文注释而导致的隐患, 中文注释前后应当再加上一行注释, 如下所示:

```
//
//检查用户名
//
if (ValidateUserName(
    document.getElementsByName("userName")[0],
    document.getElementById('lblUserName')) == false) {
    result = false ;
}
```

对于嵌入在 HTML 页面中的 JavaScript 代码则不需要这样处理。

可以使用以下方法判断 HTML、JS 或其他文本文档的编码格式。

使用二进制文档编辑器打开文件内容, 例如使用 UltraEdit 以二进制的模式显示文档内容, 若在这种模式下也能较好地显示文档中的英文内容, 则这个文档就是采用 ANSI 或 GB2312 格式存储的, 如图 12-18 所示。

若是以字节 FF 和 FE 开头的, 则说明这个文件是以 UTF-8 格式存储的, 如图 12-19 所示, 此时文档中的连续的英文字符中间还出现了大量的零字节。

若没有能以二进制查看文件内容的工具软件, 则也可以使用 Windows 记事本来进行判断。使用 Windows 记事本打开文本文件, 然后执行“另存为”操作, 弹出如图 12-20 所示的对话框。

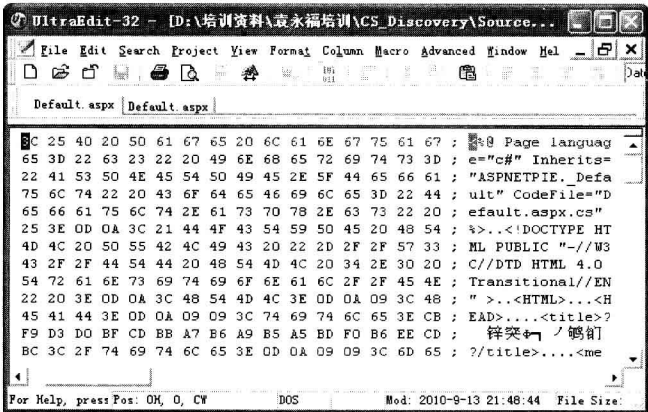


图 12-18 文档内容

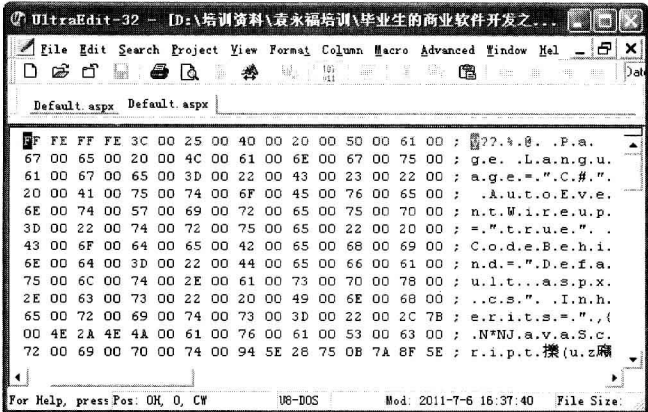


图 12-19 文档内容

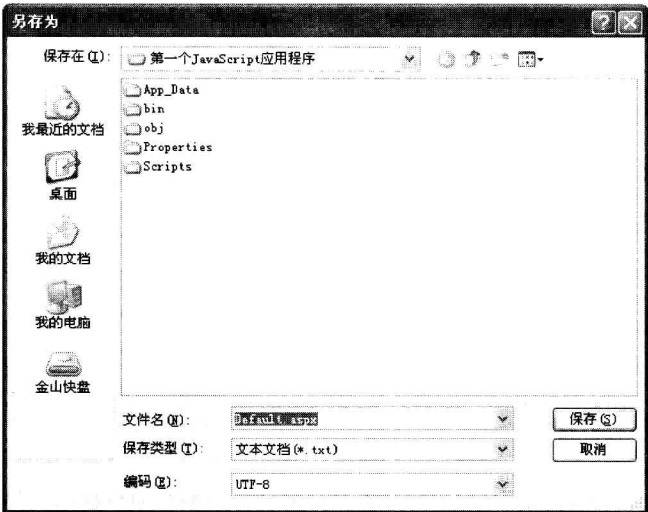


图 12-20 “另存为”对话框

在该对话框中有个“编码”下拉列表，该列表的默认值就是该文本文件的编码格式。这个编码列表中的 ANSI 编码格式放在简体中文操作系统中可以认为是 GB2312 编码格式。

特别是，当使用 Windows 记事本的“另存为”功能时，在该对话框中设置编码下拉列表就可以改变文本文件的编码格式，这样就能将 HTML、JS 或其他文本文件的编码格式在 GB2312、UTF-8、UNICODE 之间来回切换。

12.6 JavaScript 调试

使用 VS.NET 可以调试 JavaScript 代码，可以插入断点、单步调试运行。

12.6.1 设置 IE

VS.NET 只支持调试 IE 中运行的 JavaScript 代码，而且调试前首先要设置 IE 浏览器开放脚本调试功能。具体做法是，打开 IE 的“Internet 选项”对话框，如图 12-21 所示。

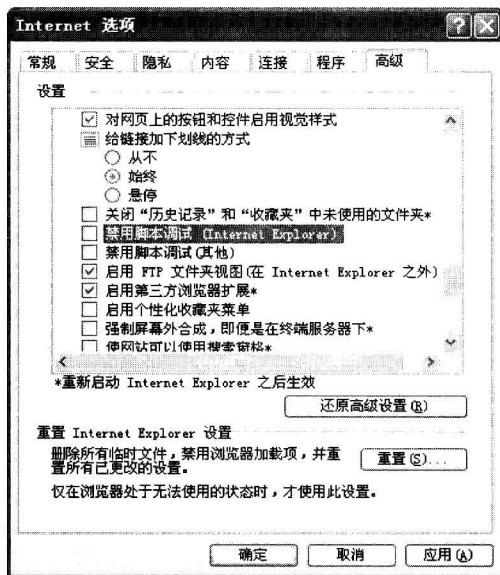


图 12-21 “Internet 选项”对话框

在该对话框中切换到“高级”选项卡，去掉“禁用脚本调试（Internet Explorer）”和“禁用脚本调试（其他）”选项前的勾选状态。

12.6.2 插入断点

可以在 JavaScript 代码中插入 debugger 语句来添加断点。当 IE 浏览器执行 JavaScript 代码遇到 debugger 语句时，就会暂停执行 JavaScript 代码，弹出如图 12-22 所示的“Visual Studio 实时

调试器”对话框。



图 12-22 “Visual Studio 实时调试器”对话框

在该对话框中，用户可以选择 VS.NET 2010，然后单击“是”按钮开始进行调试。此时系统会显示如图 12-23 所示的 VS.NET 用户界面。

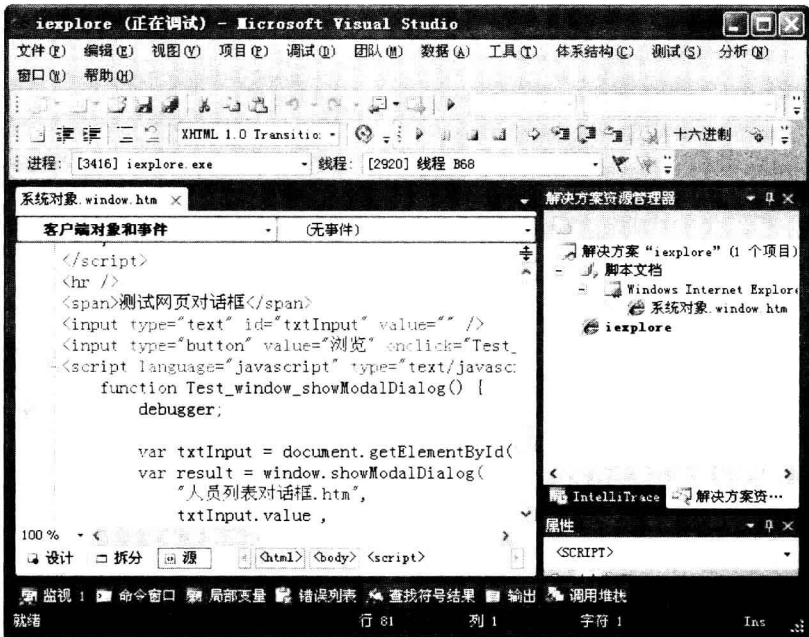


图 12-23 VS.NET 用户界面

在该界面中，VS.NET 已经打开当前显示的 HTML 页面，并突出显示出了当前执行的 JavaScript 代码，此时开发人员可以使用 VS.NET 强大的调试功能来调试 JavaScript 代码。

当 JavaScript 代码中发生错误但没有捕获时，IE 浏览器会显示如图 12-24 所示的“错误”对话框。

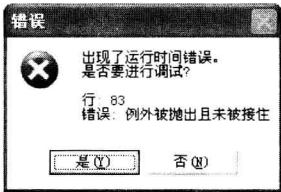


图 12-24 “错误”对话框

在该对话框中单击“是”按钮则会弹出“Visual Studio 实时调试器”对话框，让用户选择一个调试器开始进行 JavaScript 代码调试。当 VS.NET 以调试模式打开当前代码时，会显示如图 12-25 所示的对话框。



图 12-25 提示对话框

在该对话框中单击“中断”按钮，则 VS.NET 处于以调试模式进行代码单步运行的状态，而且当前执行的代码就是发生错误的代码。

12.6.3 调试

当使用 VS.NET 以调试模式打开 JavaScript 代码时，开发人员就能使用其强大的调试功能来调试 JavaScript 代码了。

开发者可以使用调试工具条上的按钮或对应的快捷键，来逐语句或逐过程地执行代码，也可以设置断点，拖曳当前执行位置，查看和修改变量值，如图 12-26 所示。

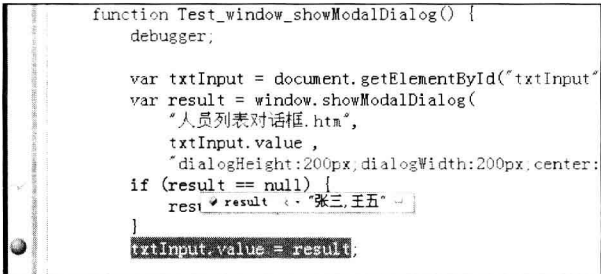


图 12-26 调试代码

但是 VS.NET 对于 JavaScript 没有实现类似 C#代码的在调试时修改代码并重新编译的功能，此时即使修改了 JavaScript 代码，虽然可以调试，但调试的还是旧代码，当前代码位置可能不对了。

12.7 JavaScript 应用实例

下面介绍一些 JavaScript 实例。

12.7.1 走马灯

JavaScript 最简单的实例就是走马灯效果，也就是在浏览器的标题栏或状态栏中以字幕的方式动态显示一段文本。

例如，以走马灯的方式在浏览器的标题栏中显示一段文本“欢迎学习袁永福的 JavaScript 技术教程”，此时浏览器标题栏将随时间动态地显示如图 12-27 所示的文本。

时刻	标题栏文本
1	欢迎学习袁永福的 JavaScript 技术教程---
2	迎学习袁永福的 JavaScript 技术教程---欢
3	学习袁永福的 JavaScript 技术教程---欢迎
4	习袁永福的 JavaScript 技术教程---欢迎学
----- 此处省略若干个时刻 -----	
20	教程---欢迎学习袁永福的 JavaScript 技术
22	程---欢迎学习袁永福的 JavaScript 技术教
23	---欢迎学习袁永福的 JavaScript 技术教程
24	-欢迎学习袁永福的 JavaScript 技术教程-
25	-欢迎学习袁永福的 JavaScript 技术教程--
26	欢迎学习袁永福的 JavaScript 技术教程---
27	----- 又开始一次循环 -----

图 12-27 浏览器标题栏显示的内容

文本“欢迎学习袁永福的 JavaScript 技术教程---”可分成 25 个部分，此时的走马灯就是循环移位显示文本“欢迎学习袁永福的 JavaScript 技术教程---”，25 个时刻完成一次循环。

为了完成走马灯功能，需要定期执行一段代码来生成文本并设置浏览器的标题栏。重点有两个，一个是定期执行代码，另一个就是生成文本。

在 JavaScript 中定期执行代码可以调用 window 全局对象的 setInterval 函数。

使用 JavaScript 生成文本也很简单，就是获得浏览器的标题文本，将第一个字符删掉，移动到后面。

以下是该走马灯效果的 JavaScript 代码。

```

var intervalID = null;

function 开始走马灯() {
    if (intervalID == null) {
        //还没有启动走马灯，启动走马灯
        document.title = "欢迎学习袁永福的 JavaScript 技术教程---";
        intervalID = window.setInterval("执行走马灯()", 200);
    }
    else {
        alert("已经启动走马灯了。");
    }
}

function 执行走马灯() {
    var text = document.title;
    var newText = text.substr(1, text.length - 1) + text.substr(0, 1);
    document.title = newText;
}

```

此处定义了一个名为 `intervalID` 的全局变量，在函数“开始走马灯”中调用了 `window` 全局对象的 `setInterval` 函数，`setInterval` 函数能定期执行一段 JavaScript 代码。第一个参数为 JavaScript 代码文本，第二个参数为以毫秒为单位的一个周期的时间间隔，函数返回值就是这个定时动作的编号。

在函数“执行走马灯”中，JavaScript 代码获得浏览器标题栏文本，然后将第一个字符移动到最后，并设置新的标题栏文本。如此就实现了走马灯效果。

使用以下 JavaScript 代码可停止走马灯。

```

function 停止走马灯() {
    if (intervalID != null) {
        window.clearInterval(intervalID);
        intervalID = null;
    }
}

```

这段代码调用 `window` 全局对象的 `clearInterval` 函数来停止定时执行，`clearInterval` 函数的参数就是 `setInterval` 函数返回的定时动作的编号。

12.7.2 网页对话框

在开发中经常用到使用对话框让用户选择输入数据的功能，此时需要使用网页对话框的功能。

实现网页对话框需要一个主界面和对话框内容页面，而且需要在主页面中调用 `window` 对象的 `showModalDialog` 函数来显示对话框，获得返回值；而在对话框内容页面中需要设置 `window` 对象的 `returnValue` 属性值来设置对话框返回值，并调用 `window` 对象的 `close` 方法自己关闭对话框。

下面演示使用网页对话框的例子，新建一个名为“人员列表对话框.htm”的 HTML 页面文件，其 HTML 内容如下：

```
<html>
<head>
    <title>请选择人员</title>
</head>
<body>
    请选择人员
    <hr />
    <input type="checkbox" name="peopleName" value="张三" />
    <span>张三</span>
    <br />
    <input type="checkbox" name="peopleName" value="李四" />
    <span>李四</span>
    <br />
    <input type="checkbox" name="peopleName" value="王五" />
    <span>王五</span>
    <br />
    <input type="checkbox" name="peopleName" value="赵六" />
    <span>赵六</span>
    <hr />
    <center>
        <input type="button" value="确定"
            onclick="OKClick()" />
        <input type="button" value="取消"
            onclick="window.close();" />
        <script language="javascript">
function OKClick() {
    //
    //收集数据
    //
    var elements = document.getElementsByName("peopleName");
    var result = "";
    for (index = 0; index < elements.length; index++) {
        if (elements[index].checked) {
            if (result.length > 0) {
                result = result + ",";
            }
            result = result + elements[index].value;
        }
    }
    //设置对话框返回值
    window.returnValue = result;
    //关闭对话框
    window.close();
}

//初始化选择状态
if (typeof (window.dialogArguments) == "string") {
    var list = window.dialogArguments;
    var items = list.split(",");
    var elements = document.getElementsByName("peopleName");
    for (index = 0; index < items.length; index++) {
        for (index2 = 0; index2 < elements.length; index2++) {
            if (elements[index2].value == items[index]) {
                elements[index2].checked = true;
                break;
            }
        }
    }
}
}
```



```

}
    </script>
  </center>
</body>
</html>

```

该页面如图 12-28 所示。

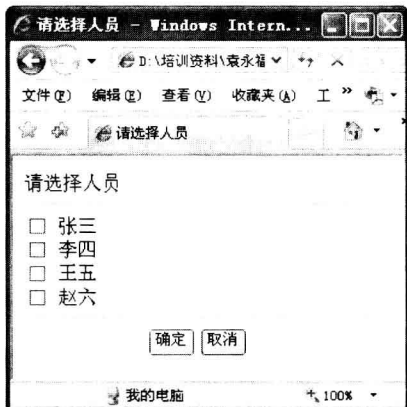


图 12-28 请选择人员

在该页面中有以下 JavaScript 脚本代码：

```

if (typeof (window.dialogArguments) == "string") {
    var list = window.dialogArguments;
    var items = list.split(",");
    var elements = document.getElementsByName("peopleName");
    for (index = 0; index < items.length; index++) {
        for (index2 = 0; index2 < elements.length; index2++) {
            if (elements[index2].value == items[index]) {
                elements[index2].checked = true;
                break;
            }
        }
    }
}
}

```

这段代码是游离在函数之外的代码，会在页面加载后自动执行。在这段代码中，首先判断 window 对象的 dialogArguments 的数据类型是否为字符串，若是字符串则进行拆分析，并设置页面中各个复选框的勾选状态。

这里还定义了以下 JavaScript 函数：

```

function OKClick() {
    //
    //收集数据
    //
    var elements = document.getElementsByName("peopleName");
    var result = "";
    for (index = 0; index < elements.length; index++) {
        if (elements[index].checked) {
            if (result.length > 0) {
                result = result + ",";
            }
        }
    }
}

```

```
        }  
        result = result + elements[index].value;  
    }  
}  
//设置对话框返回值  
window.returnValue = result;  
//关闭对话框  
window.close();  
}
```

当用户单击“确定”按钮时会执行这个函数，在函数中，代码首先收集用户选择的数据，然后设置 `window` 对象的 `returnValue` 属性值，最后调用 `window` 对象的 `close` 函数关闭本浏览器窗口。

在主页面中插入以下 HTML 代码：

```
<span>测试网页对话框</span>  
<input type="text" id="txtInput" value="" />  
<input type="button" value="浏览"  
    onclick="Test_window_showModalDialog()" />  
<script language="javascript" type="text/javascript" >  
    function Test_window_showModalDialog() {  
        var txtInput = document.getElementById("txtInput");  
  
        var result = window.showModalDialog(  
            "人员列表对话框.htm",  
            txtInput.value ,  
            "dialogHeight:200px;dialogWidth:200px;center:yes;resizable:no;  
status:no;scroll:no");  
        if (result == null) {  
            result = "";  
        }  
        txtInput.value = result;  
    }  
</script>
```

这段代码定义了 `Test_window_showModalDialog` 函数，当用户单击“浏览”按钮时，会执行这段代码。这段代码中，系统调用 `window` 对象的 `showModalDialog` 函数，函数的第一个参数是网页对话框内容页面地址，第二个参数是对话框的输入参数，第三个参数是用于控制网页对话框的一些样式。此处显示的网页对话框如图 12-29 所示。

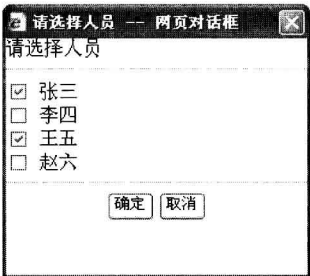


图 12-29 网页对话框

此处代码获得 `showModalDialog` 函数的返回值并显示在文本框中。

12.7.3 日历对话框

在开发中经常会让用户选择日期的数据，此时需要在用户界面上显示如图 12-30 所示的日期数据选择对话框。



图 12-30 日期数据选择对话框

该对话框具有动态效果，当用户修改当前年月时，其中的日期数也会更新。这个功能可使用 JavaScript 来实现。日历结构如图 12-31 所示。

年份下拉列表，列出从前 80 年、后 10 年的年份数				月份列表，列出了 1 月到 12 月		
<< 上月按钮		显示当前年月数			下月按钮 >>	
日	一	二	三	四	五	六
0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
小时数下拉列表，列出 1 到 24 的小时数		分钟数下拉列表		秒数下拉列表		
<div>确定按钮</div> <div>取消按钮</div>						

图 12-31 日历结构

对于这个日历，最重要的内容就是根据当前设置的年月数来设置中间的 41 个单元格的内容。下面说明其算法。

首先确定当前年和月，获得该月第一号的日期对象，即使用代码“new Date(当前年,当前月,1)”，调用它的 `getDay` 函数获得该天是星期几，然后获得该星期的星期日的日期对象。这里的

getDay 函数返回的是一个从 0 到 6 的整数，并以星期日为一个星期的第一天。

从第一个星期日的日期开始循环累加处理，循环 42 次，依次设置单元格的文本为日期对象的 getDate 函数的返回值，也就是当前日期的当月的号数。

日	一	二	三	四	五	六
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

图 12-32 日历

若正在处理的日期所在的月份数等于系统当前时间的月份数，则设置单元格的文本颜色为黑色，否则设置为灰色。若正在处理的日期的年月日数据等于系统当前时间的年月日数，则设置单元格有红色边框。此外还设置当前用户选择的日期单元格为蓝底白字。该表格的内容如图 12-32 所示。

本页面对话框的 HTML 代码比较多，此处不列出，读者可自行查看本章配套源代码。

在其他 HTML 文件中使用以下代码：

```
<input type="text" id="txtDate" value="" />
<input type="button" value="浏览" onclick="BrowseDate();" />
<script language="javascript" type="text/javascript" >
function BrowseDate() {
    var txtBox = document.getElementById("txtDate");
    var dtm = new Date(txtBox.value);
    var args = new Object();
    args.Date = dtm;
    args.ShowTime = true;
    var result = window.showModalDialog(
        "JavaScript 实例.日历.htm",
        args ,
        "dialogHeight:300px;dialogWidth:280px;resizable:no;scroll:no;
status:no");
    if (result != null) {
        txtBox.value = result ;
    }
}
</script>
```

在这段代码中，程序首先试图将文本框中的内容转换为日期数据，然后创建一个 args 的对象实例，设置它的属性值，最后将该 args 对象作为对话框输入参数传入到网页对话框中。

此处调用 window 对象的 showModalDialog 函数，并将函数返回值设置到文本框中。

12.7.4 数据验证

在 B/S 系统中存在客户端数据验证和服务器端数据验证，客户端数据验证能提高一些安全性，改善用户体验，服务器端验证能确保安全。

如果一个 B/S 系统只有客户端数据验证而没有服务器端验证，则用户输入错误的数据后能立即被告知有错，从而在提交数据前修改输入的数据。但可能有人使用技术手段跳过客户端的数据验证，从而将错误甚至导致危险的数据发往服务器端，而服务器端没有数据验证，对这些错误和危险毫无办法，任由其发作。

如果一个 B/S 系统只有服务器端验证，但没有客户端验证，此时服务器端能检查所有输入的

数据，过滤掉错误甚至导致危险的数据，确保安全。当用户无意中输入错误的数据并提交保存时，页面刷新后被告知数据错误，此时他们刚才输入的数据又得再次输入，用户体验很不好。

因此一个良好的 B/S 系统应当既具有安全性又有良好的用户体验，而 JavaScript 就能用于客户端数据验证。

JavaScript 数据验证大体分为实时数据验证和批量数据验证两种。

实时数据验证就是用户在某个输入框中输入数据，切换焦点时，JavaScript 立即验证用户输入的数据，若数据输入错误则进行提示。

批量数据验证就是用户在用户界面中已经输入所有的数据，准备提交到服务器时，JavaScript 对用户界面中所有的数据字段进行验证，若所有的数据验证都成功了则继续执行，否则中断提交数据，提示用户数据输入错误，让用户输入正确的数据后再次提交。

1. 实时数据验证

以下演示使用 JavaScript 实现实时数据验证。新建一个 HTML 页面，其正文 HTML 代码如下：

[illegible]

```
        return false;
    }
    else {
        return true;
    }
}

function ValidateEmail(element) {
    var txt = element.value;
    var reg =
/^\\w+((-\\w+)|(\\.\\w+))*@[A-Za-z0-9]+((\\.|-)[A-Za-z0-9]+)*\\. [A-Za-z0-9]+$/;
    if (txt.search(reg) >= 0) {
        return true;
    }
    else {
        alert("请输入正确格式的电子邮箱，例如'user@server.com'。");
        return false;
    }
}
</script>
</form>
```

在这个 HTML 文档中，文本输入框的 onblur 事件处理代码中都调用了 JavaScript 函数进行数据验证。例如 “onblur=“ValidateUserName(this);””，其中调用了 ValidateUserName 函数，这里的参数使用 this 表示文本输入框元素自身。

文本输入框失去输入焦点时就会触发 onblur 事件。例如对于文本框 userName，当它失去输入焦点时，触发 onblur 事件，然后调用 JavaScript 函数 ValidateUserName，若数据验证不通过，则显示一个消息框提示用户。

图 12-33 是本页面运行时的用户界面。

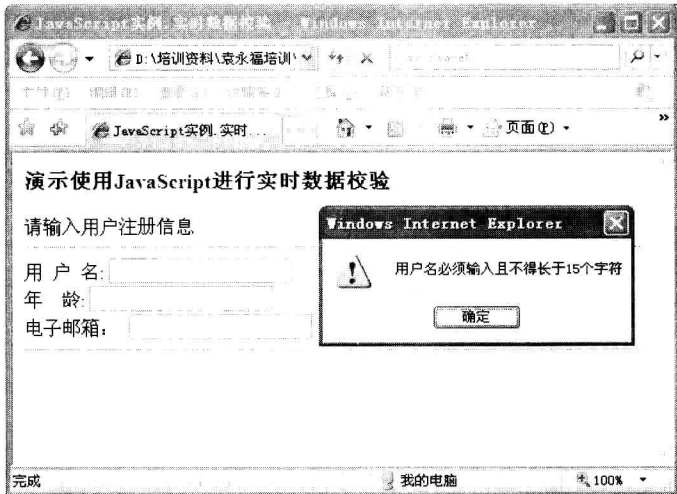


图 12-33 用户界面

其实这个页面的用户体验不太好，当来回切换输入焦点时会频繁地弹出消息框阻碍用户操作，甚至切换窗体时也会弹出消息框，让人厌烦，而且通过弹出消息框的方式不能很好地帮助用

户快速找到数据输入错误的准确位置。

为此对这个页面进行改进，使用了以下 HTML 代码：

[illegible]

```
function ValidateEmail(element, label) {
    var txt = element.value;
    var reg = /^\\w+((-\\w+)|(\\.\\w+))*\\@[A-Za-z0-9]+((\\.|-)[A-Za-z0-9]+)*\\.\\[A-Za-z0-9]+$/;
    if (txt.search(reg) >= 0) {
        element.style.backgroundColor = "";
        label.innerText = "";
        return true;
    }
    else {
        element.style.backgroundColor = "#FFCCCC";
        label.innerText = "请输入正确格式的电子邮箱，例如'user@server.com'。";
        return false;
    }
}
</script>
</form>
```

在改进后的 HTML 文档中，每个参与验证的文本框控件后面都跟着一个红色的文本标签。例如对于 userName 文本框，后面跟着 “”，它的验证数据的 JavaScript 代码为：

```
var v = element.value;
if (v.length == 0 || v.length > 15) {
    element.style.backgroundColor = "#FFCCCC";
    label.innerText = "用户名必须输入且不得长于 15 个字符";
    return false;
}
else {
    element.style.backgroundColor = "";
    label.innerText = "";
    return true;
}
```

在这段代码中，若验证成功，则设置文本框的背景色正常，后面的提示文本标签没有内容；若验证失败，则设置文本框的背景色为淡红色，而且后面会显示出提示文本。这个页面运行时的用户界面如图 12-34 所示。

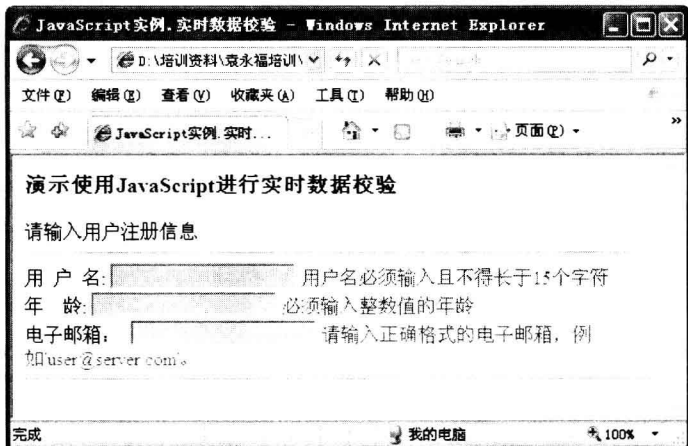


图 12-34 用户界面

这个页面的用户体验就很好了。此时以一种很人性化的、不蛮横中断用户操作的方式提醒用户数据输入错误，而且还指明了发生错误的准确位置。因此在商业软件开发中，提倡用这种客户端数据校验方式。

2. 批量数据验证

批量数据验证是指当用户输入数据时不进行数据验证，而用户准备向服务器提交数据时，才对所有的数据进行验证，验证成功则继续操作，若失败则中断操作，让用户继续输入正确的数据。以下是批量数据验证的 HTML 页面的正文内容代码。

[illegible]

```
//执行批量数据校验
```

```
function ValidateDataBatch() {
    var result = true;
    //检查用户名
    if (ValidateUserName(
        document.getElementsByName("userName")[0],
        document.getElementById('lblUserName')) == false) {
        result = false ;
    }
    //检查年龄
    if (ValidateAge(
        document.getElementsByName("userAge")[0],
        document.getElementById('lblUserAge')) == false) {
        result = false;
    }
    //检查电子信箱
    if (ValidateEMail(
        document.getElementsByName("userEmail")[0],
        document.getElementById('lblUserEmail')) == false) {
        result = false;
    }
    if (result) {
        alert("数据校验全部成功，将发送数据!");
    }
}
```

```

        else {
            alert("有数据校验失败，取消发送数据!");
        }
        return result;
    }

    function ValidateUserName(element, label) {
        var v = element.value;
        if (v.length == 0 || v.length > 15) {
            element.style.backgroundColor = "#FFCCCC";
            label.innerText = "用户名必须输入且不得长于 15 个字符";
            return false;
        }
        else {
            element.style.backgroundColor = "";
            label.innerText = "";
            return true;
        }
    }

    function ValidateAge(element, label) {
        var age = parseInt(element.value);
        if (isNaN(age)) {
            element.style.backgroundColor = "#FFCCCC";
            label.innerText = "必须输入整数值的年龄";
            return false;
        }
        if (age < 0 || age >= 150) {
            element.style.backgroundColor = "#FFCCCC";
            label.innerText = "年龄必须大于 0 而且小于 150";
            return false;
        }
        else {
            label.innerText = "";
            element.style.backgroundColor = "";
            return true;
        }
    }

    function ValidateEMail(element, label) {
        var txt = element.value;
        var reg = /^\\w+((-\\w+)|(\\.\\w+))*@[A-Za-z0-9]+((\\.|-) [A-Za-z0-9]+)*\\. [A-Za-z0-9]+$/;
        if (txt.search(reg) >= 0) {
            element.style.backgroundColor = "";
            label.innerText = "";
            return true;
        }
        else {
            element.style.backgroundColor = "#FFCCCC";
            label.innerText = "请输入正确格式的电子邮箱，例如'user@server.com'。";
            return false;
        }
    }
}
</script>
</form>

```

在这段代码中，表单元素的 `onsubmit` 事件执行了代码 “`return ValidateDataBatch();`”，而 `ValidateDataBatch` 函数对所有的文本输入框进行了验证，若全部通过则返回 `true`，否则返回

false，并在页面中显示错误信息。

表单元素的 `onsubmit` 事件是浏览器向服务器提交表单数据前一刻触发的事件，若这个事件返回 false 值，则浏览器会取消向服务器提交表单数据，否则正常提交数据。

本页面运行时的用户界面如图 12-35 所示。

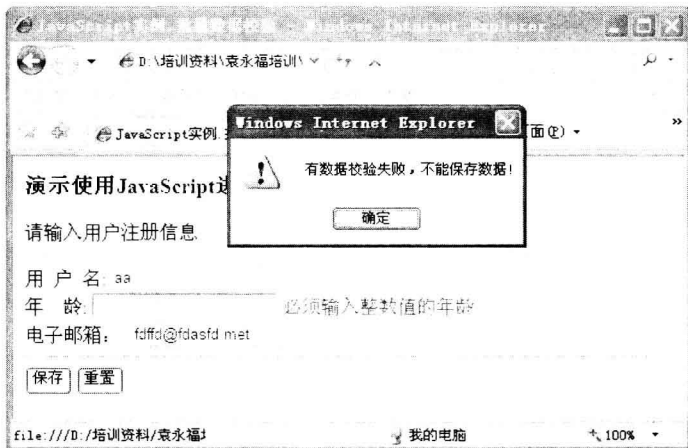


图 12-35 用户界面

在该界面中，若用户单击了“保存”按钮，但输入的数据有不对的，则会在界面中显示出数据验证错误信息，而且还弹出一个消息框明确地提示用户不能保存数据。若不弹出消息框，则出错后用户还不知道数据有没有保存好，或者以为数据已经保存成功了，这很容易造成误解，因此系统必须给予用户明示。

12.8 浏览器兼容性

JavaScript 已经是 ECMA 国际标准了，因此各种浏览器支持的 JavaScript 语法基本上是一样的。但由于历史问题，JavaScript 仍然有一些浏览器兼容性问题。也就是说，JavaScript 代码在不同的浏览器中其运行效果有可能不一样，不过这种问题相对于 HTML 来说就少多了。

JavaScript 浏览器兼容性问题很多都源自微软对 JavaScript 的扩展功能。在历史上微软为 JavaScript 添加了很多不属于国际标准的功能，但微软的 IE 浏览器占据了浏览器市场的绝大部分份额，因此纵容开发人员编写了大量的不合标准的 JavaScript，使其充斥在各种应用系统和互联网上。

现在很多其他的浏览器异军突起，比如火狐、谷歌浏览器等，这些都支持标准 JavaScript，但不支持微软开发的方言版的 JavaScript。因此很多历史遗留的 JavaScript 代码都遭遇到浏览器兼容性问题，不能在非 IE 浏览器中完全正确地运行，使得开发人员要花费一定的精力来处理浏览器兼容性问题。

还好微软的方言版 JavaScript 和标准 JavaScript 相差得也不太离谱，主要表现在访问元素属性上。

HTML 文档元素都有大量的属性，对于符合 HTML 国际标准的属性，标准的和微软版的 JavaScript 都可以直呼其名地访问其属性值。但对于不合国际标准的属性，标准 JavaScript 只能通过调用 HTML 元素对象的 `getAttribute` 方法来获得其属性值，使用 `setAttribute` 方法来设置属性值。而微软版的 JavaScript 仍然可以直呼其名地访问其属性值。

比如对于 HTML 代码 “`sohu 首页`”，HTML 元素 “a” 的属性 “id” 和 “href” 是符合 HTML 国际标准的。因此在标准的及微软版的 JavaScript 中都可以使用 “`link.id`” 或 “`link.href`”，来获得或设置属性值。对于 “tag” 属性其不符合 HTML 国际标准，因此在标准的 JavaScript 中不能使用 “`link.tag`” 来获得或设置其属性值，只能使用 “`link.getAttribute("tag")`” 或 “`link.setAttribute("tag")`” 来访问其属性值。但在微软版的 JavaScript 中 “`link.tag`” 是有效的，而且也支持 “`getAttribute`” 或 “`setAttribute`” 方法来访问属性值。

12.9 小结

本章主要讲述 JavaScript 语言的原理、语法以及简单应用。相信读者通过本章的初步学习就可以使用 JavaScript 技术了。

JavaScript 是所有 Web 开发的重要基础技术，任何 Web 开发者，包括 C#、Java 或 PHP 等，都需要学习和掌握 JavaScript 技术。JavaScript 的水平高低也是 Web 开发者水平高低的一个重要指标。

开发第一个 XML 应用程序

本章将介绍 XML 的基本知识，说明 XML 技术的历史、基本内容以及在 IT 行业中的使用情况，并演示如何在 ASP.NET 应用程序中使用 XML 技术。

13.1 XML 应用框架

XML 技术是近年来最重要的软件开发技术，无论是 C# 和 Java 还是其他软件技术都没有它重要，它已经和 HTML 技术一起成为现在 Web 开发的基础，而且比 HTML 更具有活力，还在不断开拓更广泛的应用领域。目前对于 Web 开发人员，XML 技术的应用框架如图 13-1 所示。

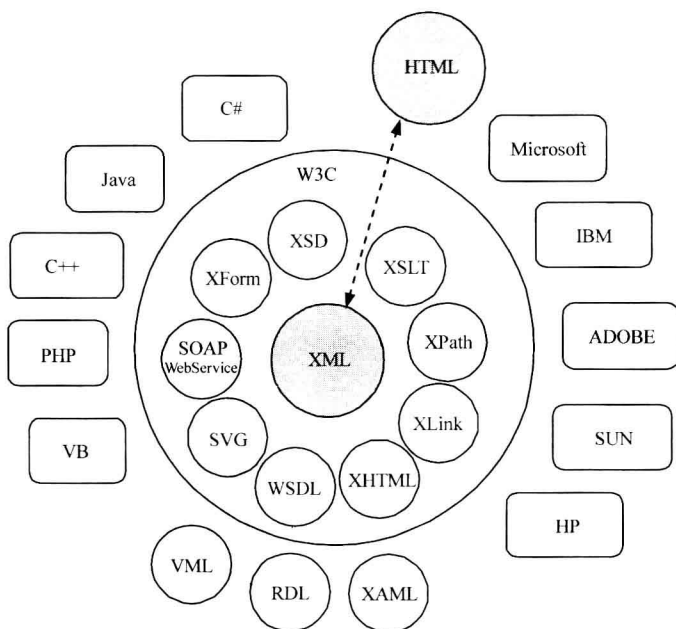


图 13-1 XML 应用框架

XML 技术是一种国际标准，受 W3C 国际标准组织的控制，并在其基础上派生了 XSLT、XSD、XPath、XHTML 等国际标准技术，并且和 HTML 关系密切。大量的业界公司也积极支持

XML 技术，并参与其中的标准制定。各家公司在 XML 的基础上又制定了一些非国际标准的 XML 应用，比如微软就制定了 VML、RDL 和 XAML 的 XML 应用。此外大量的编程语言，包括 Java、C#、C++等都强力支持 XML 技术。可以看到以 XML 技术为核心，周围聚集了大量的开发资源。

13.2 XML 的发展历史

XML 是一种使用纯文本标记语法来描述复杂树状数据结构的语法。这项技术的起源最早可追溯到 20 世纪 60 年代。

图 13-2 是 XML 标准的发展历史演示图。

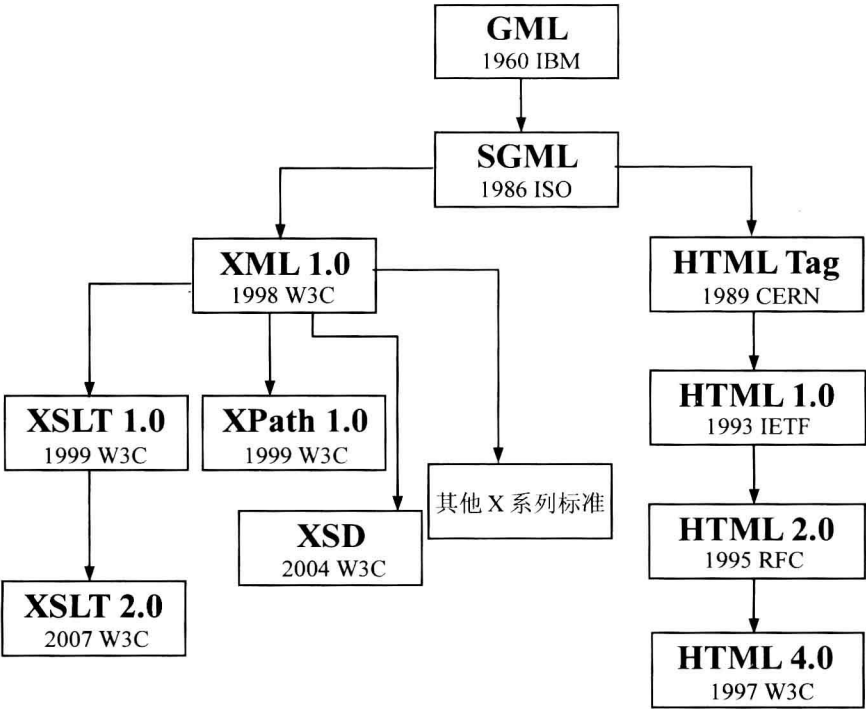


图 13-2 XML 发展历史演示图

为了促进各个信息化系统之间的数据交换和操作，在 20 世纪 60 年代，IBM 公司开始发展一种叫做 GML (Generalized Markup Language, 通用标记语言) 的标记语言，形成一种通用的文档格式。GML 是一种格式化文档语言，用于对文档的组织结构、各部分及其之间的关系进行描述。GML 将这些描述标记为章节、重要小结、次要小结、段落、列表、表格等。

1978 年，ANSI (美国国家标准学会) 将 GML 加以整理规范，发布形成 SGML，1986 年被 ISO (国际标准化组织) 采用形成 ISO 8879 标准，并得到广泛引用。SGML 是一种与语言无关

的、结构化的、可扩展的标记语言，这些特点使得很多组织和企业使用它来创建、处理和发布大量的文本信息。但 SGML 内容非常严谨，过于庞大复杂，难于理解和学习，这影响了其推广使用。

1989 年，CERN 欧洲粒子物理研究中心的研究人员基于 SGML 开发的超文本标记语言，成为 HTML (Hyper Text Markup Language)。HTML 简单实用，并被一些国际组织在其初期没有任何定义文档外观的相关内容，仅仅用于在浏览器中显示带样式的文档内容。后来随着因特网的发展，人们为 HTML 扩展了样式控制的内容。1993 年，IETF (因特网工程工作小组) 发布了 HTML 1.0 的工作草案 (并非标准)，1995 年，RFC1866 发布了 HTML 2.0 标准，1997 年，W3C 发布了 HTML 4.0 标准。

后来人们发现 HTML 的一些缺陷，特别是语法不够严谨，扩展性不好，不大适合用于保存纯粹的数据。为此 1996 年，W3C 组织开始在 SGML 的基础上进行精简，并参考 HTML 的发展经验，制定了一套使用上规则严谨但内容简单的描述数据的标记语言，形成 XML。1998 年，W3C 制定了 XML1.0 的推荐标准。后来 W3C 和其他国际标准组织在 XML 的基础上制定了很多适用于特定应用的衍生标准。

13.3 XML 基础知识介绍

13.3.1 XML 基本语法知识

XML 全名是可扩展标记语言，是 W3C 国际标准组织规定的一种基于纯文本的数据存储格式。它是从 IBM 的 SGML 技术派生的，可以说 XML 使用了 SGML 的 20% 的语法实现了 SGML 的 80% 的功能。

从软件开发人员的角度看，XML 语法的主要内容有：

(1) XML 是国际标准，绝大部分软件厂商、开发工具和编程语言都支持相同的基本 XML 语法。XML 技术可用于任何开发平台上，移植成本低，这是 XML 技术最大的优势之一。

(2) XML 技术是基于纯文本的，XML 文档中不能包含二进制数据，而且存储 XML 文件时会涉及文本编码格式的问题。对于中国软件开发人员，存储 XML 文档时使用的文本编码格式一般有 unicode、utf-8 和 gb2312。使用错误的文本编码格式加载和保存 XML 文档会出现中文乱码问题。

(3) XML 是区分大小写的，所有的 XML 元素名称、XML 属性名称都区分大小写。

(4) XML 文档具有层次结构，其中使用一对尖括号来定义一个 XML 元素，一个 XML 元素可以包含若干个属性，而 XML 元素下面又可以包含若干个子 XML 节点，节点名称区分大小写。XML 语法检查非常严格，语法比 HTML 严谨得多。

(5) 一个 XML 文档只能而且必须定义一个根元素，不可以多定义，也不能不定义。

(6) XML 元素不能错乱套嵌定义，比如 “<a>” 是错误的 XML 文档，而应该定义为 “<a>”。

(7) XML 格式是为了各信息系统交流数据而设计的，其设计过程考虑了方便数据的临时存储和交流，而不考虑数据的长期存储。因此 XML 文档比较冗余，文件体积大，不适合存储大批量数据，在计算机网络中使用 XML 格式传输数据效率比较低。在软件开发中需要注意到这个问题。

另外 XML 支持 CDATA 文本标记，用于以可视化的方式容纳带有大量尖括号的文本，例如有以下一段文本：

```
<html xmlns='http://www.w3.org/1999/xhtml'>
<head runat='server'>
  <title></title>
</head>
<body>
  <form id='form1' runat='server'>
    <div>

    </div>
  </form>
</body>
</html>
```

若以一般方式存储在 XML 文档中，则存储代码为：

```
<Element>
&lt;html xmlns='http://www.w3.org/1999/xhtml'&gt;
&lt;head runat='server'&gt;
  &lt;title&gt;&lt;/title&gt;
&lt;/head&gt;
&lt;body&gt;
  &lt;form id='form1' runat='server'&gt;
    &lt;div&gt;

    &lt;/div&gt;
  &lt;/form&gt;
&lt;/body&gt;
&lt;/html&gt;
</Element>
```

这种方式使人们难于阅读和修改。

若这段文本以 CDATA 方式存储在 XML 文档中，则存储代码为：

```
<Element><![CDATA]
<html xmlns='http://www.w3.org/1999/xhtml'>
<head runat='server'>
  <title></title>
</head>
<body>
  <form id='form1' runat='server'>
    <div>

    </div>
  </form>
</body>
</html>
```


]]></Element>

用这种方式人们就能很方便地阅读和修改了。

13.3.2 W3C 国际标准组织

此处多次提到 W3C 国际标准组织，那么到底什么是 W3C 国际标准组织呢？

W3C 是一些重要的软件企业联合起来制定某些重要软件业标准的国际组织。它的成员包括微软，IBM，SUN 等软件巨头。它制定和维护了 HTML、XHTML、HTTP、XML、VML、XPath、XForm 等软件行业内重要的标准，绝大多数软件厂商都支持 W3C 制定的标准。它制定的标准是真正的跨平台的全球通用的，因此它对全球软件行业，尤其是 Web 软件行业有着巨大的影响。它的网址是 <http://www.w3c.org>，在它的网站上可以看到它所制定的上百个标准。软件开发人员若要开发具有国际水平的 Web 应用系统，首先要好好学习 W3C 的某些标准。

13.3.3 国际标准的意义

所谓国际标准就是指某个权威的非营利性的国际组织，其立场中立，不代表某个具体的公司，而代表整个业界，它针对某项普遍使用的技术出台一些规范和标准，而绝大多数软件厂商在运用这项技术时都自觉遵守这套国际标准。这样能方便各个系统之间交流数据，保障异构系统进行集成，并保持数据结构的长期稳定性和兼容性。这样的国际组织有 ISO，ECMA 和 W3C 等。

我们使用到的一些技术都已经成为国际标准，例如 SQL，JavaScript，C#，HTML，XML，XSLT，HTTP 等。

国际标准具有一些特点，首先是稳定性和连贯性，国际标准一旦正式发布，就保持了相当的稳定性，其内容只能慎重地增加而尽量不删减。国际标准组织不会轻易修改已经正式发布的国际标准，而且在修改标准时会充分考虑到各种因素，保证向上和向下的兼容性，能最大程度地保障业界在旧标准上的投资。这些国际组织发布国际标准时有时会事先提出标准的修订计划。

其次国际标准是全球业界都遵守的，虽然没有强制遵守的机制，但绝大多数软件厂商都会遵守或努力遵守这些国际标准。而且国际标准组织的成员有很多是大软件厂商，比如 W3C 的成员就有微软，IBM，SUN 等大公司。因此国际标准代表了广大软件业界的根本利益，代表了最先进的软件生产力。

对于应用软件开发商，充分运用国际标准能在很大程度上保护客户在 IT 系统上的投资。由于国际标准具有相当的稳定性和连贯性，若客户 IT 系统充分地使用了这些国际标准，则在升级到新标准时能获得很好的兼容性，IT 系统不用推倒重来，这样能保护客户在已有系统上的投资。

作为软件开发人员，应当了解这些国际标准，能比较容易地实现异构系统的集成，并能获得比较好的系统兼容性和可维护性；而且软件开发人员在切换开发平台，比如从 Java 转移到 .NET 平台上时，以前学习国际标准的投资就会得到保护，而遵守相同国际标准的源代码的移植和翻译也是低成本的。

13.4 微软.NET 框架对 XML 的支持

微软.NET 框架提供了对 XML 的强大支持，而且.NET 框架本身也普遍采用 XML 格式来存储各种配置信息，比如 web.config 文件。

在.NET 类库中，名称空间 System.Xml 下面就包含了大量的操作 XML 文档的类型。这些类型构成了两种 XML 文档的处理模型。

13.4.1 流式处理模型

在流式处理模型中，程序将 XML 文档当做一个数据流来进行处理，将逐个处理 XML 文档中的数据。在这种模型下，程序可以以只读的方式快速读取大体积的 XML 文档，而且内存占用少，程序性能好。类型 System.Xml.XmlReader 就提供了流式处理模型，使用 XmlReader 可以快速读取 XML 文档。

使用流式处理模型是有缺点的，首先它只能读取 XML 文档，不能修改 XML 文档；其次检索 XML 文档内容不方便，不能使用 XPath 技术；而且编程接口比较简单，处理 XML 文档不够方便。当程序需要比较简单地从 XML 文档读取数据时，可以采用流式处理模型。

13.4.2 DOM 处理模型

在 DOM 处理模型中，程序首先用文档对象模型的思想解析整个 XML 文档，然后在内存中生成一个对象树来表述 XML 文档。比如使用一个 XmlElement 对象来影射 XML 文档中的一个元素，使用 XmlAttribute 对象来影射 XML 文档中的一个属性。这样开发人员编程操作内存中的对象就影射为操作 XML 文档。

使用 DOM 处理 XML 文档具有相当大的优点，首先是处理方便，开发人员可以使用各种编程技巧来处理 XML 文档对象树状结构，比如可以递归遍历 XML 文档的一部分或全部，可以向树状结构插入、修改或删除 XML 元素，可以设置 XML 元素的属性。

在 DOM 模式下，开发人员可以使用 XPath 技术在 XML 文档树状结构中进行快速检索和定位，这为处理 XML 文档带来比较大的方便。

在 C# 中，开发人员可以很简单地使用 DOM 方式处理 XML 文档。首先实例化一个 System.Xml.XmlDocument 类型，调用它的 Load 方法加载 XML 文档并生成 XML 节点对象树状结构，然后就可以遍历这个对象树，新增、修改和删除节点，而且其中的任意一个节点都可以使用 SelectNodes 或 SelectSingleNode 方法通过 XPath 相对路径快速查找其他的节点。

在名称空间 System.Xml 下面大部分类型都是用来支持 DOM 处理模型的。其中很多类型配合起来共同组成 XML DOM，XML DOM 是一种很典型的对文档对象模型的应用。

System.Xml 名称空间下支持 DOM 的类型主要有以下 10 种（见图 13-3）。

（1）XmlNode 是 DOM 结构中所有类型的基础类型，它定义了所有 XML 节点的通用属性和方法，是 XML DOM 的基础。它具有一个 ChildNodes 属性，表示它所包含的子 XML 节点。

（2）XmlAttribute 表示 XML 属性，它只保存在 XmlElement 的 Attributes 列表中。

（3）XmlDocument 表示 XML 文档本身，是 XML DOM 模型中的顶级对象，它用于对 XML 文档进行整体的控制，并且是其他程序访问 XML 文档对象树的唯一入口。

（4）XmlLinkedNode 在 XmlNode 的基础上实现了访问前后同级节点的方法。

（5）XmlElement 元素表示 XML 元素，是 XML DOM 中使用最多的对象类型。它具有 Attributes 属性，可以处理它所拥有的属性，可以使用 ChildNodes 属性获得它所有的子节点，并提供了一些添加和删除子节点的方法。

（6）XmlCharacterData 表示 XML 文档中的字符数据的基础类型。字符文本数据是分布在各个 XmlElement 之间的纯文本数据。XmlAttribute 中的文本数据是不属于 XML 文本块的。

（7）XmlCDataSection 表示 XML 文档中的 CDATA 节，CDATA 数据是用“<![CDATA[...]]>”括起来的纯文本数据。由于 XML 采用尖括号进行标记，因此具有和 HTML 类似的转义字符。在一般的 XML 纯文本段中，当遇到尖括号等特殊字符时，需要使用转义字符，当文本段中包含大量的这类特殊字符时，手工书写和阅读 XML 文档将比较困难，为了改善 XML 文档的可读性，在此可以使用 CDATA 节。在 CDATA 节中，所有的字符，包括特殊字符都不需要转义，这样查看和修改 XML 文档都比较方便。

（8）XmlComment 表示一段注释，XML 注释和 HTML 注释一样，使用一对“<!--...-->”来括起来。

（9）XmlText 表示一段纯文本数据。

（10）XmlWhitespace 表示 XML 文档中一段纯粹由空白字符组成的独立文本块，空白字符包括空格、制表符、换行和回车符，全角空格不属于空白字符。XmlDocument 在解析 XML 文档时会处理空白字符，当 XmlDocument 对象的 PreserveWhiitespace 属性为 true 时，会为 XML 文档中的纯空白文本块生成 XmlWhitespace 对象，若该属性为 false，则会忽略纯空白文本，不会生成 XmlWhitespace 对象，就像原始的 XML 文档中不存在这样的空白文本块一样。

在 XmlDocument 对象的 Load 函数中，系统会解析 XML 文档的文本内容，会根据其中的各个部分创建一个 XML 文档节点对象，并按照 XML 文档中的层次结构将这些 XML 节点对象组织成一个树状列表，形成 XML 文档对象，其结果如图 13-4 所示。

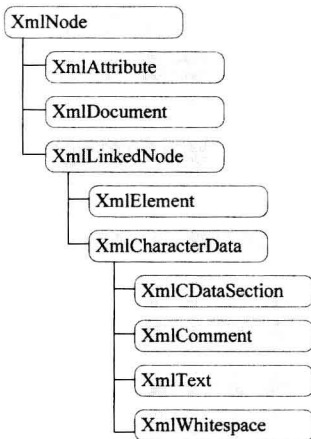


图 13-3 10 种类型

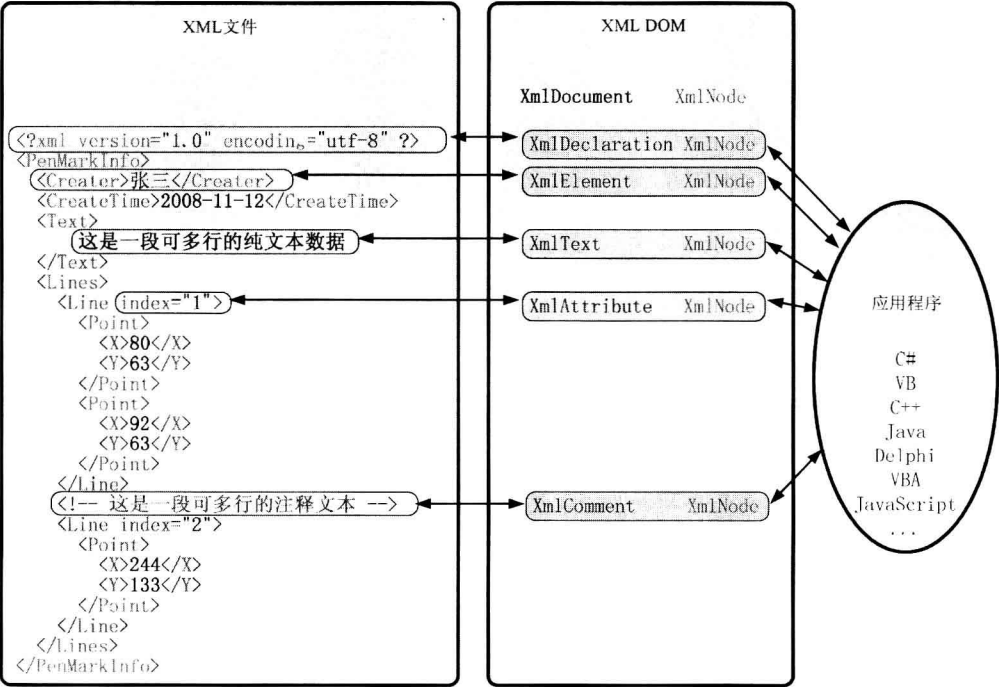


图 13-4 形成 XML 文档对象的过程

可以通过操作 XML 文档对象来间接地读取和修改 XML 文档的内容和结构，大大降低处理结构复杂的 XML 文档的难度，使得 XML 技术得以普及。这里使用了一种称为“文档对象模型”的软件设计模式，它是一种比较高级的软件设计模式。

13.5 输出 XML 文档

下面演示在 ASP.NET 应用程序中输出 XML 文档。

在演示数据库 Customers.mdb 中有一张数据表“人员基本信息表”，表 13-1 是该数据表中的一些数据，此处并不显示所有的记录和所有的字段。

表 13-1 人员基本信息表中的一些数据

RecordID	所属单位 ID	姓 名	部 门	职 位	电 话	传 真
1	2	周伯通	市场部	销售员	(030) 355576470	0
2	2	游坦之	市场部	销售代表	(069) 202459840	(069) 202458740
3	5	黄牛	市场部	销售代表	(025) 305984100	(025) 305985110
4	0	萧何	市场部	物主	(030) 311234560	(030) 311335570
5	5	林冲	市场部	销售经理	(030) 617761100	(030) 617761110

续表

RecordID	所属单位 ID	姓 名	部 门	职 位	电 话	传 真
6	6	包拯	市场部	销售员	(030) 296725420	(030) 296733330
7	5	李世民	市场部	采购员	(0241) 103912310	(0241) 105942820
8	0	慕容博	市场部	销售代理	(0217) 20124670	(0217) 20124680
9	3	无名	市场部	销售经理	(0571) 862132430	(0571) 862233440

在 VS.NET 中，新建一个名为“第一个 XML 应用程序”的 ASP.NET Web 应用程序。该应用程序将读取上面的数据并生成 XML 文档展现出来。

将数据库 Demo.mdb 复制到工程所在目录的 App_Data 子目录下，并包含在工程中。

13.5.1 PageUseXmlTextWriter.aspx

新建一个 PageUseXmlTextWriter.aspx 的页面。该页面将使用流的方式生成 XML 文档。打开文件 UseXmlTextWriter.aspx，删除其中所有的 HTML 代码，只保留第一行代码，注意该行最后的看不见的换行回车符也要删除，则该 ASPX 文件的内容如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="PageUseXmlText-
Writer.aspx.cs" Inherits="第一个 XML 应用程序.PageUseXmlTextWriter" %>
```

在 ASPX 页面的 HTML 代码中，除引用 C#代码外就没有任何其他内容了，因此该页面的所有内容都是用 C#代码生成的。

XML 文档是一个纯文本，因此理论上可以使用字符串拼凑来生成 XML 文档，但实际开发中不建议使用字符串拼凑来生成 XML 文档，而应采用 XmlTextWriter 或 XmlDocument 对象来生成并输出 XML 文档。

打开 PageUseXmlTextWriter 页面的 C#代码，在该页面的 Page_Load 函数里添加以下 C#代码：

```
//此处使用 XmlTextWriter 来快速输出 XML 文档内容，不构造 XML 文档对象结构
this.Response.ContentEncoding = System.Text.Encoding.GetEncoding(936);
this.Response.ContentType = "text/xml";
//连接数据库
using (IDbConnection conn = new OleDbConnection())
{
    conn.ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + this.Server.MapPath("App_Data\\Demo.mdb");
    conn.Open();

    //查询数据库
    using (IDbCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = @"
            Select RecordID , 姓名 , 部门 , 职位 , 电话,传真
            From 人员基本信息表
            Order By 姓名";
        IDataReader reader = cmd.ExecuteReader();

        //获得所有字段名
```

```
int FieldCount = reader.FieldCount;
string[] FieldNames = new string[FieldCount];
for (int iCount = 0; iCount < FieldCount; iCount++)
{
    FieldNames[iCount] = reader.GetName(iCount);
}

//基于 HTTP 输出流生成一个 XML 文档书写器
System.Xml.XmlTextWriter myXmlWriter = new System.Xml.XmlTextWriter(
    this.Response.Output);
//设置缩进格式
myXmlWriter.Indentation = 3;
myXmlWriter.IndentChar = ' ';
myXmlWriter.Formatting = System.Xml.Formatting.Indented;

//开始输出 XML 文档
myXmlWriter.WriteStartDocument();

myXmlWriter.WriteStartElement("Records");
while (reader.Read())
{
    //输出一条记录
    myXmlWriter.WriteStartElement("人员基本信息");
    for (int iCount = 0; iCount < FieldCount; iCount++)
    {
        //输出一个字段值
        myXmlWriter.WriteStartElement(FieldNames[iCount]);
        object v = reader.GetValue(iCount);
        if (v == null || DBNull.Value.Equals(v))
        {
            myXmlWriter.WriteAttributeString("Null", "1");
        }
        else
        {
            myXmlWriter.WriteString(Convert.ToString(v));
        }
        myXmlWriter.WriteEndElement();
    }
    myXmlWriter.WriteEndElement();
} //while( reader.Read())
reader.Close();
//结束输出 XML 文档
myXmlWriter.WriteEndElement();
myXmlWriter.WriteEndDocument();
myXmlWriter.Close();

} //using
} //using
```

下面介绍这段 C# 代码的执行流程。

1. 设置 HTTP 输出类型

首先设置 HTTP 文档的输出类型，程序设置输出的文本编码格式为“GB2312”，此处使用代码“GetEncoding(936)”就是获得 GB2312 的编码格式。

代码还设置 HTTP 输出对象的 ContentType 属性来设置文档的输出格式。在 HTTP 协议中，

ContentType 属性描述了文档输出类型，当文档传递到客户端时，客户端浏览器获得 ContentType 属性值，查询注册表和 Windows 中注册的 COM 信息，获得该属性值确定的文件类型，然后使用相应的模式显示文档。

例如，设置 ContentType 属性为 “application/vnd.ms-excel”，则客户端浏览器查询注册表得知，对应的文件类型信息在注册项目 “HKEY_CLASSES_ROOT\.xls” 下面，本地文件类型为 “Excel.Sheet.8”，如图 13-5 所示。然后又根据其他信息转而调用 EXCEL 的 COM 组件来显示获得的 HTTP 文档。

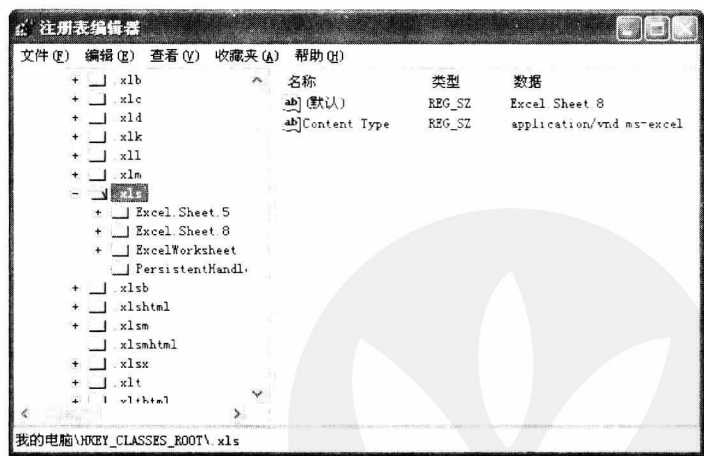


图 13-5 注册表编辑器

从 ContentType 属性的说明我们可以知道，要想很深入地掌握 Web 开发，有时候是需要了解一些 Windows 编程知识的。因为 B/S 系统的客户端就是各种浏览器，而 IE 浏览器就是比较复杂的 Windows 程序。

2. 查询数据库

设置了 HTTP 文档输出模式后，程序开始输出 XML 文档内容。首先是连接数据库，使用演示数据库 Demo.mdb，执行一个 SQL 查询 “Select RecordID, 姓名, 部门, 职位, 电话, 传真 From 人员基本信息表 Order By 姓名 Select * From Customers”，获得一个数据读取器对象 “reader”。

3. 使用 XmlTextWriter 输出 XML 文档

查询得到数据后，程序就可以遍历查询所得的数据库记录，开始输出 XML 文档，此处程序使用 XmlTextWriter 输出文档。

要输出 XML 文档程序有两种选择，一种是使用 XmlTextWriter 输出，另一种是从 XmlDocument 类型开始构造 XML 文档对象结构，然后使用 XmlDocument 的 Save 方法输出 XML 文档。两种方法有各自的特点。

使用 `XmlTextWriter` 可以以只写的向前的方式快速输出 XML 文档，而且输出时不能获得已经输出的 XML 文档内容，也不能修改已经生成的 XML 文档。这种方法速度快，占用内存少，但不够灵活。

而使用 `XmlDocument` 类型构造 XML 文档结构后，再输出 XML 文档的方法比较灵活，程序可以随时访问和修改已经输出的 XML 文档。这种方法速度慢，占用内存多，但很灵活。

在这里程序要使用 `XmlTextWriter` 来输出 XML 文档，在另外的一个页面使用 `XmlDocument` 输出 XML 文档。

程序首先在 HTTP 页面输出流上创建一个 `XmlTextWriter` 对象，设置它的启动缩进样式。它的 `Indentation`、`IndentChar` 和 `Formatting` 属性共同控制 XML 文档缩进样式，`Indentation` 属性设置每层的缩进量，`IndentChar` 属性设置缩进使用的字符，一般为空格或制表符，`Formatting` 属性用于启动或禁止缩进样式，关于这三个属性的具体说明可以参考 MSDN。XML 文档缩进样式是为了改善 XML 文档的可读性，带有缩进样式的 XML 文档便于人们直接阅读和修改，但对应用程序来说，XML 文档是否有缩进是没多大差别的。

`XmlTextWriter` 类型是一个基于其他流的针对输出 XML 文档的包装，它本身不能打开文件，因此在初始化 `XmlTextWriter` 时必须指明其内部使用的文档内容输出对象。文档内容输出对象可以为流（类型 `System.IO.Stream` 的派生类）或文本书写器（类型 `System.IO.TextWriter` 的派生类）。

由于 XML 文档和 HTML 文档一样，是纯文本文档，因此理论上开发人员可以使用字符串拼凑来生成 XML 文档，但实际开发中使用字符串拼凑 XML 文档不是明智之举，建议使用 `XmlTextWriter`。由于 HTML 文档没有很严格的语法限制，IE 浏览器能解释劣质的 HTML 代码，因此会有开发者进行这样的字符串拼凑操作来生成 HTML 文档，但这会使代码比较杂乱，可读性不好。

而 XML 文档具有很严格的语法检查，只要有一个 XML 语法错误就会导致整个 XML 文档解析错误。因此开发人员应当使用 `XmlTextWriter`，因为它能帮助程序实现基本的 XML 语法，确保程序能输出合格的 XML 文档。

程序调用 `XmlTextWriter` 类型的 `WriteStartDocument` 方法来开始输出 XML 文档，并调用 `XmlEndDocument` 方法来结束输出 XML 文档。

`XmlTextWriter` 类型提供了很多配对使用的成员方法，使用一个方法后需要使用另外一个配对方法。比如 `WriteStartDocument` 和 `WriteEndDocument` 配对，`WriteStartElement` 和 `WriteEndElement` 配对，配对的方法必须成对调用。此处程序使用 `WriteStartDocument` 开始书写 XML 文档，最后必须使用 `WriteEndDocument` 来结束输出 XML 文档。在使用 `XmlTextWriter` 输出 XML 文档的时候，`WriteStartDocument` 必须是第一个调用的方法。

程序调用 `WriteStartElement` 方法来输出 XML 文档的根节点，这里使用的函数参数值为字符串“Records”，表示输出的 XML 文档的根节点名称为“Records”。

接着程序使用数据库数据读取器的“Read”函数来遍历所有查询的数据。对于每一条记录，使用 XML 书写器的 WriteStartElement 方法来输出 XML 元素，这里参数为字符串“人员基本信息”，表示输出的 XML 元素名为“人员基本信息”，而且这个节点被添加到 XML 文档的根节点下。

对于每一条记录程序还遍历其所有的字段值，对每一个字段值使用 WriteStartElement 函数新增一个 XML 元素，元素名称就是各个字段的名称。若字段值为空则使用 WriteAttributeString 输出名为“NULL”的 XML 属性，否则使用 WriteString 来输出字段值的字符串形式。

由于 WriteStartElement 方法和 WriteEndElement 方法配对使用，因此每输出完一个 XML 元素后，都需要调用 WriteEndElement 方法来结束输出 XML 元素。当所有的内容输出完毕后，程序调用 WriteEndDocument 方法来结束输出整个 XML 文档。

4. 测试运行该页面

PageUseXmlTextWriter.aspx 页面的运行界面如图 13-6 所示。



图 13-6 运行界面

从该运行界面中可以看出，客户端的 IE 浏览器识别出 PageUseXmlTextWriter.aspx 页面返回的文档是 XML 格式的，并以缩进高亮度的方式显示了该 XML 文档。该页面的源代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<Records>
  <人员基本信息>
    <RecordID>56</RecordID>
    <姓名>包龙星</姓名>
    <部门>销售部</部门>
    <职位>市场经理</职位>
    <电话>(0571) 203345600</电话>
    <传真>(0571) 203345610</传真>
  </人员基本信息>
  <人员基本信息>
    <RecordID>6</RecordID>
    <姓名>包拯</姓名>
    <部门>市场部</部门>
    <职位>销售员</职位>
    <电话>(030) 296725420</电话>
  </人员基本信息>
</Records>
```

```
</人员基本信息>
<人员基本信息>
  <RecordID>6</RecordID>
  <姓名>包拯</姓名>
  <部门>市场部</部门>
  <职位>销售员</职位>
  <电话>(030) 296725420</电话>
  <传真>(030) 296733330</传真>
</人员基本信息>
<人员基本信息>
..... 其他内容 .....
</人员基本信息>
</Records>
```

13.5.2 PageUseXmlDocument.aspx

上一个页面是使用 XmlTextWriter 类型来输出 XML 文档的。下面介绍使用 XmlDocument 对象以 DOM 处理模型的方式来输出 XML 文档。

新建一个名为“PageUseXmlDocument.aspx”的页面，删除 ASPX 文件中的所有 HTML 代码，只保留第一行，然后在该页面的 Page_Load 方法中输入以下 C# 代码：

```
//此处代码动态构造 XmlDocument 对象来输出 XML 文档
System.Xml.XmlDocument XmlDoc = new System.Xml.XmlDocument();
XmlDoc.AppendChild(XmlDoc.CreateElement("Records"));

//连接数据库
using (IDbConnection conn = new OleDbConnection())
{
    conn.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + this.Server.MapPath("App_Data\\Demo.mdb");
    conn.Open();

    //查询数据库
    using (IDbCommand cmd = conn.CreateCommand())
    {
        cmd.CommandText = @"
            Select RecordID , 姓名 , 部门 , 职位 , 电话,传真
            From 人员基本信息表
            Order By 姓名";
        IDataReader reader = cmd.ExecuteReader();

        //获得所有字段名
        int FieldCount = reader.FieldCount;
        string[] FieldNames = new string[FieldCount];
        for (int iCount = 0; iCount < FieldCount; iCount++)
        {
            FieldNames[iCount] = reader.GetName(iCount);
        }

        while (reader.Read())
        {
            //输出一条记录
            System.Xml.XmlElement RecordElement
```

```

        = XmlDoc.CreateElement("人员基本信息");
        XmlDoc.DocumentElement.AppendChild(RecordElement);
        for (int iCount = 0; iCount < FieldCount; iCount++)
        {
            //输出一个字段值
            System.Xml.XmlElement FieldElement
                = XmlDoc.CreateElement(FieldNames[iCount]);
            RecordElement.AppendChild(FieldElement);
            object v = reader.GetValue(iCount);
            if (v == null || DBNull.Value.Equals(v))
            {
                FieldElement.SetAttribute("Null", "1");
            }
            else
            {
                FieldElement.AppendChild(
                    XmlDoc.CreateTextNode(Convert.ToString(v)));
            }
        }
    } //while( reader.Read())
    reader.Close();
} //using
} //using

//输出生成的 XML 文档
XmlDoc.Save(this.Response.Output);

```

在这段代码中，程序首先创建一个 `XmlDocument` 对象实例，`XmlDocument` 实例创建时表示一个空的 XML 文档，没有任何内容，没有根元素，因此第一步就是使用 `XmlDocument` 对象的 `AppendChild` 方法来添加根元素。在这里我们使用了 `XmlDocument` 对象的 `CreateElement` 函数来创建一个名为“Records”的 `XMLElement` 元素对象，然后调用 `AppendChild` 方法添加到 XML 文档中。

各种类型的 XML 文档对象，包括元素、属性、文本块、注释等，都不能直接实例化，只能使用 `XmlDocument` 对象的一系列以 `Create` 开头的函数来创建对象实例；而且创建的 XML 文档对象是一个个离散的对象，必须及时地添加到 XML 文档对象结构中，才能真正成为 XML 文档的一部分。一般使用 XML 文档对象或元素对象的 `AppendChild` 方法将新创建的 XML 文档对象添加到指定对象下面，这样才能加入 XML 文档结构的大家庭中。

这种处理模式类似向 `DataTable` 对象添加新的数据行。`DataRow` 对象本身不能直接实例化，首先得使用 `DataTable` 对象的 `NewRow` 方法创建一个新的 `DataRow` 对象实例，然后再使用 `DataTable` 对象的 `Rows` 属性的 `Add` 方法向数据表添加刚刚创建的数据行。

初始化一个 XML 文档对象后，程序连接数据库，查询数据库获得一个数据读取器，然后遍历查询所得的数据库记录，输出 XML 文档。

对每一个数据库记录，首先创建一个 `RecordElement` 对象，添加到 XML 文档的根节点下；然后遍历数据库记录的每一个字段值，创建一个 `FieldElement` 对象并添加到 `RecordElement` 下面；若当前数据库字段值为空，则调用 `FieldElement` 的 `SetAttribute` 方法，设置名为 `Null` 的属性值为 1，否则向 `FieldElement` 添加一个 XML 文本节点。

完成生成 XML 文档后，程序调用 XML 文档对象的 Save 方法向 HTTP 页面输出流输出 XML 文档的内容。这里的 Save 方法将以 UTF-8 的文本编码格式输出 XML 文档内容。

运行本演示程序，打开 UseXmlDocument.aspx 页面，可以看到其内容和使用 XmlTextWriter 对象输出 XML 文档的 UseXmlTextWriter.aspx 页面的显示效果是一样的，只是 XML 文本编码格式不一样。

13.6 类型使用参考说明

在本程序中使用了 System.Xml.XmlTextWriter 等类型，下面进行说明。

13.6.1 System.Xml.XmlTextWriter 类型

XmlTextWriter 类型提供快速的、不缓存的、只能输出不能修改的 XML 文档书写器，使用该类型生成的 XML 文档能符合最基本的 XML 规范。

该类型常用的属性如表 13-2 所示。

表 13-2 常用属性

属 性 名 称	属 性 类 型	说 明
BaseStream	System.IO.Stream	基础流对象。XmlTextWriter 只是在其他流之上做了一个包装，本身不能容纳数据，必须由一个基础流接受数据
Formatting	System.Xml.Formatting	指明是否对输出的 XML 文档进行缩减处理，属性值为一个枚举类型，可选值有： None：不进行格式化，这是默认选项 Indented：进行缩进处理
Indentation	int	表示输出的 XML 文档中每个层次之间的缩进量，默认为 2
IndentChar	char	表示缩减 XML 文档使用的字符，默认为空格

XmlTextWriter 类型常用的方法如表 13-3 所示。

表 13-3 常用方法

方 法 签 名	说 明
public XmlTextWriter(System.IO.TextWriter)	构造函数，根据一个文本书写器初始化对象
public XmlTextWriter(System.IO.Stream,System.Text.Encoding)	构造函数，使用指定的数据输出流和文本编码格式初始化对象
public XmlTextWriter(string ,System.Text.Encoding)	构造函数，使用指定的文件名和文本编码格式初始化对象
void Close()	关闭对象，关闭基础流
void Flush()	将内部缓存区中的数据刷新到基础流中，并刷新基础流本身。 XmlTextWriter 内部有一个很小的缓存区用来提高性能

续表

方 法 签 名	说 明
void WriteAttributeString(string , string)	输出 XML 属性，第一个参数为属性名，第二个参数为属性值
void WriteBase64(byte[] bs , int index ,int count)	将指定的二进制数据编码为 Base64 的文本并输出
void WriteCDATA(string)	输出包含指定文本的<![CDATA[...]]> 块
void WriteComment(string)	写出包含指定文本的注释 <!--...-->
void WriteElementString(string , string)	输出包含指定文本的元素。第一个参数为元素名称，第二个参数为元素包含的纯文本值
void WriteProcessingInstruction(string , string)	写出在名称和文本之间带有空格的处理指令，如<?name text?>。第一个参数为指令名称，第二个参数为指令文本
void WriteRaw(string)	原封不动地输出文本。注意系统不会检查要输入的值是否符合 XML 语法，因此这样有可能输出不合法的 XML 文档
void WriteStartDocument() void WriteEndDocument()	这是一对作用相反的方法，用于开始和结束输出 XML 文档。调用 XmlTextWriter 对象输出 XML 文档时，在输出任何内容前首先要调用一次 WriteStartDocument 方法，完成输出所有的内容后，必须调用 WriteEndDocument 方法。也就是说，在 XmlTextWriter 对象诸多 Write 开头的方法中，WriteStartDocument 是首先调用的，WriteEndDocument 是最后一次调用的
void WriteStartElement(string) void WriteEndElement()	这是一对作用相反的方法，用于开始和结束输出一个 XML 元素
void WriteStartAttribute(string) void WriteEndAttribute()	这是一对作用相反的方法，用于开始和结束输出一个 XML 属性
void WriteString(string)	输出 XML 文本数据
void WriteValue(object)	输出数值
void WriteWhitespace(string)	输出空白文本

13.6.2 System.Xml.XmlDocument 类型

XmlDocument 类型是以 DOM 方式处理 XML 文档的核心类型，它表示一个 XML 文档，而且是应用程序访问 XML DOM 树状结构的入口。

该类型常用的公开属性如表 13-4 所示。

表 13-4 常用的公开属性

属 性 名	数 据 类 型	说 明
DocumentElement	XmlElement	返回根节点对象
InnerText	string	返回 XML 文档的文本值
InnerXml	string	返回 XML 文档中的 XML 值
PreserveWhitespace	bool	指示是否在元素内容中保留空白，默认为 false

XmlDocumnet 类型常用的公开方法如表 13-5 所示。

表 13-5 常用的公开方法

方 法 签 名	说 明
XmlNode AppendChild(XmlNode)	添加节点到文档最后
XmlAttribute CreateAttribute(string)	创建具有指定名称的 XML 属性对象
XmlCDataSection CreateCDataSection(string)	创建具有指定内容的 XML 文本块对象
XmlComment CreateComment(string)	创建指定内容的注释对象
XmlElement CreateElement(string)	创建指定名称的 XML 元素对象
XmlText CreateTextNode(string)	创建指定内容的文本对象
XmlWhitespace CreateWhitespace(string)	创建指定内容的空白文本对象
XmlElement GetElementById(string)	获得具有指定 ID 值的 XML 元素对象
XmlNodeList GetElementsByTagName(string)	获得文档中由所有指定名称的 XML 节点对象组成的列表
void Load(System.IO.Stream) void Load(string) void Load(System.IO.TextReader) void Load(System.Xml.XmlReader)	从指定的来源加载 XML 文档，解析其中的内容并生成 DOM 对象树。参数可以是流对象、文件名或文本读取器
void LoadXml(string)	加载分析指定的 XML 字符串，并生成 DOM 对象树
void Save(System.IO.Stream) void Save(string) void Save(System.IO.TextWriter) void Save(System.Xml.XmlWriter)	保存 XML 文档
XmlNodeList SelectNodes(string)	执行指定的 XPATH 查询，返回由参数所得的节点组成的列表
XmlNode SelectSingleNode(string)	执行指定的 XPATH 查询，返回第一个查询所得的节点对象

13.6.3 System.Xml.XmlElement 类型

XmlElement 类型表示 XML 文档中的一个元素。它可以包含子元素，可以有若干个属性。该类型常用的属性如表 13-6 所示。

表 13-6 常用属性

名 称	数 据 类 型	说 明
Attributes	XmlAttributeCollection	XML 属性列表
HasAttributes	bool	判断对象是否有属性
InnerText	string	对象下所有的文本值
InnerXml	string	设置或获得表示对象子节点的 XML 文本
Name	string	只读属性，获得元素的名称
NextSibling	XmlNode	获得 XML 文档中紧接着该元素的 XML 节点对象
OwnerDocument	XmlDocument	获得元素对象所属的 XML 文档对象
ParentNode	XmlNode	父节点对象

XmlElement 类型常用的公开方法如表 13-7 所示。

表 13-7 常用的公开方法

方法 签 名	说 明
XmlNode CloneNode(bool)	复制节点。参数用于指明是否复制子孙节点
string GetAttribute(string)	获得指定名称的属性值，若没有指定名称的属性则方法返回空字符串
bool HasAttribute(string)	判断是否存在指定名称的属性
void RemoveAll()	删除该节点所有的属性和子节点
void RemoveAllAttributes()	删除所有的属性
void RemoveAttribute(string)	删除指定名称的属性
XmlNodeList SelectNodes(string)	以本节点为出发点，执行 XPATH 查询，并返回由查询所得的 XML 节点组成的列表
XmlNode SelectSingleNode(string)	以本节点为出发点，执行 XPATH 查询，并返回查询所得的第一个节点
void SetAttribute(string , string)	设置属性值。第一个参数为属性名，第二个参数为属性值
void WriteContentTo(XmlWriter)	将本节点下的所有子节点内容输出到指定的 XML 书写器中
void WriteTo(XmlWriter)	将本节点以及所有的子节点的内容输出到指定的 XML 书写器中

13.7 小结

本章简单介绍了 XML 技术的一些基本知识，说明了处理 XML 文档使用的流式处理模式和 DOM 处理模式，并演示如何使用 XmlTextWriter 对象和 XmlDocument 对象在 ASP.NET 中查询数据库并输出 XML 文档。

XML 是一项不简单的技术，而且在其上面又派生了很多其他技术。作为当代的软件开发人员，尤其是 Web 开发人员，应当熟练掌握和使用 XML 技术及其某些派生技术。熟悉 XML 技术有助于开发者长期保持相当水平的软件开发能力，也是学习其他技术的重要基础。

开发第一个文件系统操作应用程序

虽然 ADO.NET 提供强大的功能使得应用程序能在数据库中存储各种数据，但访问本地文件系统，在文件中存储数据还是很必要的，有很多应用程序都有这个需求。在本章中将演示如何开发第一个文件系统操作应用程序。

14.1 文件系统操作概述

任何操作系统都提供文件管理功能，Windows 操作系统也提供了大量的 API 接口让应用程序能访问本地文件。

文件系统具有以下基本概念。

14.1.1 文件和目录

文件系统中有两种对象，即文件对象和目录对象。文件是存储数据的，目录是用于组织管理文件系统的，一个目录可以包含若干个文件和若干个子目录，子目录还可以包含下一层子目录。同一个目录下，文件或子目录的名称不能重复，但不同目录下，文件或子目录可以重名。

文件和目录有很多属性，比如文件名、创建时间、修改时间、最后访问时间、是否是系统文件、是否隐藏等，对于文件还有文件长度属性。

14.1.2 文件路径

文件路径用于说明文件在文件系统中的定位名称，它是唯一的，两个文件的路径不会重复。文件路径包含磁盘符、目录名称、文件名三个部分。

比如对于文件路径“D:\培训资料\袁永福培训\毕业生的商业软件开发之路.vsd”，其中的“D:”说明这个文件是存储在计算机中 D 盘的；“培训资料\袁永福培训”是这个文件的目录路径，其中“培训资料”是根目录，“袁永福培训”是其下的子目录，各级目录名之间用斜杠分开；“毕业生的商业软件开发之路”是文件名；“.vsd”是该文件的扩展名。

在 Windows 系统中，文件路径和文件名长度最多不超过 255 个字符，而且不能包含特殊字符“\ / : * ? \ " < > | ”。

14.1.3 文本文件和二进制文件

文件根据其存储的内容大体可分为文本文件和二进制文件两种。

文本文件就是指文件内容是纯文本的，可以用记事本打开而不会出现乱码，人们可以直接阅读文件内容。例如，对于扩展名为 txt、xml、html、log 的文件有可能是文本文件。

二进制文件就是指文件内容不是纯文本的，而是二进制格式的，用记事本打开会出现乱码，人们不能直接阅读和理解文件内容。例如，对于扩展名为 exe、dll、doc、xls 的文件有可能是二进制文件。

14.1.4 .NET 的文件系统开发

微软.NET 框架对文件系统提供了强大的支持，.NET 类库中命名空间 System.IO 下面的类型就是用于支持文件系统开发的。该命名空间下主要有如表 14-1 所示的 7 种类型用来访问文件系统。

表 14-1 命名空间 System.IO 下面的类型

类 型	功 能
DriveInfo	能获得计算机中所有磁盘的信息，如盘符、磁盘类型、磁盘大小、可用空间大小等
DirectoryInfo	能获得目录的名称、创建时间等信息，还能获得所有文件和子目录的信息
FileInfo	能获得文件的名称、创建信息、长度等信息，还能提供文件读写功能
FileStream	提供以二进制格式读写文件的功能
Path	提供对路径的一些功能例程，比如获得文件路径名所在的目录名，扩展名等
StreamReader	提供以文本方式读取文件内容的功能
StreamWriter	提供以文本方式保存文件内容的功能

使用这些类型，应用程序就能比较完整地操作文件系统。需要注意的是，.NET 框架的文件系统接口都受.NET 框架安全控制，在 Windows 桌面程序中基本上没有什么限制，但在 ASP.NET 或 IE 智能客户端中程序操作文件系统会受到不小的限制，因此在编写程序时需要遵守最小权限的原则。

14.2 建立 C#应用程序项目

使用 VS.NET 2010 新建一个名为“第一个文件系统操作应用程序”，类型为“Windwos 窗体应用程序”的 C#项目。下面开始开发这个文件系统操作应用程序。

14.2.1 设计主窗体

在解决方案资源管理器中修改默认建立的“Form1.cs”的窗体文件名为“frmMain.cs”，在正常情况下，系统会将该文件中的类型 Form1 修改成 frmMain。双击“frmMain.cs”节点，打开该

窗体的设计界面。

在属性编辑器中修改该窗体的 Text 属性值为“文件系统管理器”，然后从左边的工具箱中拖曳一个“ToolStrip”控件到窗体上。

ToolStrip 控件是一种工具条控件，可以放置若干个按钮，其设计界面如图 14-1 所示。

在窗体设计器中，用鼠标单击选中刚刚添加的 ToolTip 控件，此时会在该控件左边显示一个小按钮，单击该按钮会弹出如图 14-2 所示的菜单。

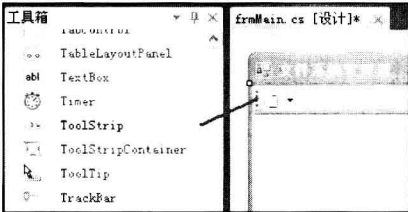


图 14-1 设计界面

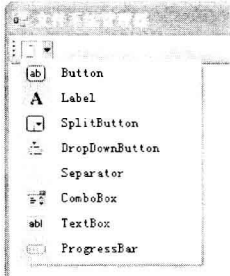


图 14-2 菜单

通过该菜单可以在工具条控件上添加各种按钮。该菜单项目列出的按钮类型如表 14-2 所示。

表 14-2 按钮类型

按钮类型	说明
Button	功能按钮，可以显示图标和文字。用户可以按下该按钮调用程序功能
Label	静态文本，只能显示一个文本，用户不能对它进行操作
SplitButton	下拉菜单按钮，可以显示图标和文字，用户单击它会显示一个下拉菜单
Separator	分隔栏，仅仅显示一个无功能的分隔栏，用于对同一个工具条上的按钮分组
ComboBox	下拉列表，可在工具条上放置一个下拉列表控件，可直接输入文本，也可通过下拉列表选择项目
TextBox	文本框。可在工具条上放置一个可直接输入的文本框
ProgressBar	进度条

单击菜单项目“Button”，可在工具条上添加一个按钮，用鼠标单击选择这个按钮，在右边的属性列表中设置该按钮的属性值如表 14-3 所示。

表 14-3 按钮的属性值

属性名称	值	说明
(Name)	btnRefresh	控件的名称，同一个窗体中控件名不能重复
AutoToolTip	False	是否显示提示文本。由于这里直接显示了按钮文本，因此不必显示提示文本
DisplayStyle	ImageAndText	按钮显示方式，该属性值可设置为以下 5 种类型： None：按钮不显示任何内容，如 <input type="checkbox"/> Text：只显示文本，如 加载... Image：只显示图标，如 ImageAndText：既显示文本又显示图标，如 加载...

续表

属 性 名 称	值	说 明
Text	刷新	按钮中显示的文本。在 Windows 用户界面中，若按钮或菜单文本后面有三个小点，表示单击该按钮或菜单后会弹出一个对话框，这是一个惯例，应当遵守

继续进行类似的操作，在工具条上再添加几个按钮，此时对工具条的设计如图 14-3 所示。其中“打开文件”按钮名为“btnOpen”，“关闭”按钮名为“btnClose”，而且“关闭”按钮的 Aligment 属性值为 Right。

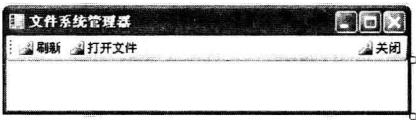


图 14-3 文件系统管理器

工具条设计好后，从工具箱中拖曳一个“TreeView”控件到窗体上，设置该控件的属性值如表 14-4 所示。

表 14-4 控件的属性值

属 性	属 性 值
(Name)	twvFileSystem
Dock	Left
HideSelection	False

从工具箱上拖曳一个“Splitter”控件到窗体上，再拖曳一个“ListView”控件到窗体上，设置这个控件的属性值如表 14-5 所示。

表 14-5 控件的属性值

属 性 名	属 性 值
(Name)	lvwFileSystem
Columns	在弹出的数据栏目对话框中添加“名称”、“大小”、“修改日期”等栏目
Dock	Fill
FullRowSelect	True
GridLines	True
HideSelection	False
View	Details

调整 lvwFileSystem 控件中各个栏目的宽度，这样主窗体用户界面的设计如图 14-4 所示。

在该用户界面中，左边为一个树状列表控件，用于显示目录树；右边为一个视图列表控件，用于显示当前目录下的文件和子目录信息。

14.2.2 浏览目录

在图 14-4 中，左边的树状列表列出了本地文件系统中的目录树状结构，其中第一层为磁盘，以下为各个磁盘中的目录结构。由于文件系统中可能存在成千上万个目录和文件，因此不能

加载所有的文件目录结构到这个树状列表中，必须采用动态加载，也就是在需要的时候逐层加载目录结构。

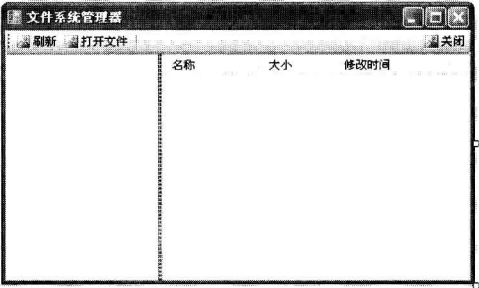


图 14-4 主窗体设计器

在图 14-4 中，双击“刷新”按钮，则系统切换到该窗体的 C#代码编辑界面，并自动生成“刷新”按钮的点击事件处理方法的代码，代码如下：

```
private void btnRefresh_Click(object sender, EventArgs e)
{
    tvwFileSystem.BeginUpdate();
    this.Cursor = Cursors.WaitCursor;
    tvwFileSystem.Nodes.Clear();
    foreach (DriveInfo d in DriveInfo.GetDrives())
    {
        TreeNode rootNode = CreateDirecotryNode(d.RootDirectory);
        tvwFileSystem.Nodes.Add(rootNode);
    }
    tvwFileSystem.EndUpdate();
    this.Cursor = Cursors.Default;
}

private object loadingFlag = new object();
























private TreeNode CreateDirecotryNode(DirectoryInfo dir)
{
    TreeNode node = new TreeNode(dir.Name);
    node.Tag = dir;
    TreeNode node2 = new TreeNode("正在加载...");
    node2.Tag = loadingFlag;
    node.Nodes.Add(node2);
    return node;
}
```

在按钮事件处理方法 btnRefresh_Click 中，代码“tvwFileSystem.BeginUpdate()”表示开始以较大的动作更新这个树状列表内容，暂时停止重新绘制内容。若没有这段代码，对树状列表进行很大的更新动作时，会添加或删除几百上千的节点，则这个过程会比较缓慢，因为每更新一个节点都会导致树状列表的重新绘制。而调用 BeginUpdate 方法是为了通知树状列表控件暂时不需要绘制用户界面，这样在大范围地操作树状列表的节点时速度不会慢。






代码“this.Cursor = Cursors.WaitCursor”是设置窗体的鼠标光标形式为等待样式的，这样让用户知道程序正在执行操作，需要等待。

类型 `System.Windows.Forms.Cursors` 中定义了多个静态属性，这些属性的值定义了各种系统预定义的鼠标光标，这些鼠标光标的样式如表 14-6 所示。

表 14-6 鼠标光标的样式

Cursors 类型的静态属性	说 明	光标样式
AppStarting	获取在应用程序启动时显示的光标	
Arrow	获取箭头光标	
Cross	获取十字线光标	
Default	获取默认光标，它通常为箭头光标	
Hand	获取手形光标，当悬停在 Web 链接上时通常使用该光标	
Help	获取“帮助”光标，它是箭头和问号的组合	
HSplit	获取当鼠标定位在水平拆分条上时显示的光标	
IBeam	该光标用于显示单击鼠标时文本光标出现的位置	
No	获取指示某个特定区域对当前操作无效的光标	
NoMove2D	获取滚轮操作过程中，鼠标不动但是该窗口可以同时水平和垂直方向上滚动时显示的光标	
NoMoveHoriz	获取滚轮操作过程中，鼠标不动但是该窗口可以在水平方向上滚动时显示的光标	
NoMoveVert	获取滚轮操作过程中，鼠标不动但是该窗口可以在垂直方向上滚动时显示的光标	
PanEast	获取滚轮操作过程中，鼠标移动并且该窗口水平向右滚动时显示的光标	
PanNE	获取滚轮操作过程中，鼠标移动并且该窗口水平、垂直向上和向右滚动时显示的光标	
PanNorth	获取滚轮操作过程中，鼠标移动并且该窗口垂直向上滚动时显示的光标	
PanNW	获取滚轮操作过程中，鼠标移动并且该窗口水平、垂直向上和向左滚动时显示的光标	
PanSE	获取滚轮操作过程中，鼠标移动并且该窗口水平、垂直向下和向右滚动时显示的光标	
PanSouth	获取滚轮操作过程中，鼠标移动并且该窗口垂直向下滚动时显示的光标	
PanSW	获取滚轮操作过程中，鼠标移动并且该窗口水平、垂直向下和向左滚动时显示的光标	
PanWest	获取滚轮操作过程中，鼠标移动并且该窗口水平向左滚动时显示的光标	
SizeAll	获取四个方向大小调整光标，该光标由相连接的、分别指向东南西北的四个箭头组成	
SizeNESW	获取双向对角线（东北/西南）大小调整光标	
SizeNS	获取双向垂直（北/南）大小调整光标	

续表

Cursors 类型的静态属性	说 明	光标样式
SizeNWSE	获取双向对角线（西北/东南）大小调整光标	
SizeWE	获取双向水平（西/东）大小调整光标	
UpArrow	获取上箭头光标，该光标通常用来标识插入点	
VSplit	获取当鼠标定位在垂直拆分条上时显示的光标	
WaitCursor	获取等待光标，通常为沙漏形状	

在不同的计算机中，随着不同的操作系统版本或用户设置，其鼠标光标样式会有所不同。

在 Windows 窗体应用程序中，需要根据程序运行状态来设置窗体的鼠标光标样式，这样能帮助用户理解程序的状态，改善用户体验。

在“刷新”按钮的点击事件处理方法中，代码“`tvwFileSystem.Node.Clear()`”用于清空树状列表控件中所有的节点。

代码“`DriveInfo.GetDrives()`”用于获得计算机系统中所有逻辑磁盘的信息对象数组，在这个方法中程序遍历所有的逻辑磁盘信息并向树状列表中添加根节点。

处理完所有的逻辑磁盘后，执行代码“`tvwFileSystem.EndUpdate()`”以消除此前代码“`tvwFileSystem.BeginUpdate()`”的效果，允许控件重新绘制其内容。

代码“`this.Cursor = Cursors.Default`”用于设置窗体的鼠标光标状态为正常，这样能帮助用户知道操作已完成。这段代码还定义了“`CreateDirectoryNode`”方法，该方法能为一个目录对象创建一个树状列表节点对象。在该方法中，根据目录名称创建一个节点对象，并设置节点对象的 Tag 属性值为目录信息对象，然后为该节点创建一个文本为“正在加载...”的子节点用于占位。

单击“刷新”按钮后，程序的用户界面如图 14-5 所示。

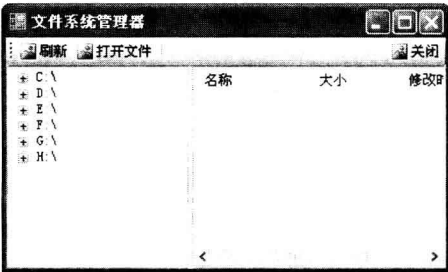
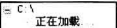


图 14-5 用户界面

在该用户界面中，双击展开节点“C:\”，则该节点显示为。这是由于加载该节点是为了添加“正在加载...”子节点，而按理应该列出该目录下所有的子目录。因此当第一次展开某个目录节点需要为其加载子目录节点时，需要处理树状列表控件的 `AfterExpand` 事件来添加子目录节点。在窗体设计器中选树状列表控件，在属性编辑器中切换到事件列表，双击 `AfterExpand` 事件，则系统打开该窗体的 C# 代码窗口，并自动生成“`tvwFileSystem_AfterExpand`”方法。需要添加的代码如下：

```
private void tvwFileSystem_AfterExpand(object sender, TreeViewEventArgs e)
{
    LoadChildNodes(e.Node);
}

private void LoadChildNodes(TreeNode rootNode)
{
    if (rootNode.FirstNode != null
        && rootNode.FirstNode.Tag == loadingFlag)
    {
        //加载子目录
        rootNode.Nodes.Clear();
        DirectoryInfo dir = (DirectoryInfo)rootNode.Tag;
        foreach (DirectoryInfo subDir in dir.GetDirectories())
        {
            TreeNode node = CreateDirecotryNode(subDir);
            rootNode.Nodes.Add(node);
        }
    }
}
```

在这段代码中，定义了 LoadChildNodes 方法，该方法就是为了获得子目录生成子节点。其中代码“dir.GetDirectories()”可获得指定目录下的所有子目录信息。

添加这段代码后，程序能无限地显示目录结构，程序运行时的用户界面如图 14-6 所示。



图 14-6 用户界面

在该用户界面中，左边的树状列表控件已经能完整地显示出本地文件系统目录结构。

14.2.3 浏览文件

程序主界面右边的视图列表，需要列出树状列表中当前节点表示的目录下的所有子目录和文件。因此还需要处理树状列表控件的 AfterSelect 事件，代码如下：

```
private void tvwFileSystem_AfterSelect(object sender, TreeViewEventArgs e)
{
    lvwFileSystem.BeginUpdate();
    lvwFileSystem.Items.Clear();
}
```

```
        if (tvwFileSystem.SelectedNode != null)
        {
            DirectoryInfo rootDir = (DirectoryInfo)tvwFileSystem.SelectedNode.
Tag;
            //首先列出所有的子目录
            foreach (DirectoryInfo dir in rootDir.GetDirectories())
            {
                ListViewItem item = new ListViewItem(dir.Name);
                item.SubItems.Add("");
                item.SubItems.Add(dir.LastWriteTime.ToString("yyyy-MM-dd HH:
mm:ss"));
                item.Tag = dir;
                lvwFileSystem.Items.Add(item);
            }
            //列出所有的文件
            foreach (FileInfo file in rootDir.GetFiles())
            {
                ListViewItem item = new ListViewItem(file.Name);
                item.SubItems.Add(file.Length.ToString());
                item.SubItems.Add(file.LastWriteTime.ToString("yyyy-MM-dd HH:
mm:ss"));
                item.Tag = file;
                lvwFileSystem.Items.Add(item);
            }
        }
        lvwFileSystem.EndUpdate();
    }
```

在这段代码中，首先遍历当前目录下的子目录，针对每一个子目录创建一个 ListViewItem 对象实例并添加到视图列表控件中；然后遍历所有的文件，针对每一个文件创建一个 ListViewItem 对象实例并添加到视图列表控件中。

该视图列表控件第一栏显示的是文件或目录的名称，第二栏是文件大小，第三栏是文件或目录的最后修改时间。由于目录没有大小的概念，因此第二栏为空。文件信息对象 FileInfo 的 Length 属性就是文件大小；文件或目录信息对象的 LastWriteTime 属性就是它的最后访问时间。代码“LastWriteTime.ToString("yyyy-MM-dd HH:mm:ss")”中的“yyyy-MM-dd HH:mm:ss”用于指明时间数据转换为字符串时的格式。

经过上述开发，本程序运行时的用户界面如图 14-7 所示。

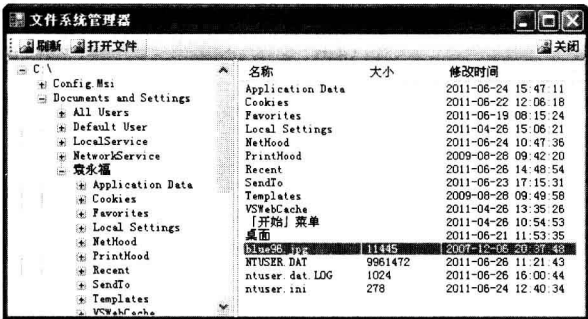


图 14-7 用户界面

这样文件系统管理器的浏览文件系统结构的功能已经设计完了。

14.2.4 查看、编辑文本内容

下面需要实现查看和编辑文本文件内容的功能。新建一个名为“frmEdit”的窗体，设置其属性如表 14-7 所示。

表 14-7 frmEdit 窗体的属性

属 性	属 性 值
MinimizeBox	False
StartPosition	CenterScreen
ShowInTaskbar	False
Text	文本编辑

在窗体上放置一个 TextBox 控件，设置其属性如表 14-8 所示。

表 14-8 TextBox 控件的属性

属 性	属 性 值
(Name)	txtContent
AcceptsTab	True，本属性用于设置让文本框允许接受 Tab 按键字符。若本属性值设置为 True，则用户在文本框获得焦点的情况下按下 Tab 键会在文本框中输入一个制表符；若该属性值为 False，则用户按下 Tab 键会将输入焦点切换到下一个控件
Anchor	Top, Bottom, Left, Right
Multiline	True，允许输入和显示多行文本
ScrollBars	Both
WordWrap	False

在窗体上添加一个 Button 控件，设置其属性如表 14-9 所示。

表 14-9 添加的 Button 控件的属性值

属 性	属 性 值
(Name)	btnSave
Anchor	Bottom,Right
Text	保存(&S)

再添加一个 Button 控件，设置其属性如表 14-10 所示。

表 14-10 再次添加的 Button 控件的属性值

属 性	属 性 值
(Name)	btnClose
Anchor	Bottom,Right
Text	关闭(&C)

窗体的设计如图 14-8 所示。



图 14-8 窗体设计界面

按下快捷键 F7 显示该窗体的 C# 代码，在窗体对象的构造函数中添加一行代码“this.DialogResult = System.Windows.Forms.DialogResult.Cancel;”。其构造函数全部代码如下：

```
public frmEdit()
{
    InitializeComponent();
    this.DialogResult = System.Windows.Forms.DialogResult.Cancel;
}
```

在这个窗体的 C# 代码中添加一个 FileName 的公开属性，其代码为：

```
private string _FileName = null;
/// <summary>
/// 处理的文件名
/// </summary>
public string FileName
{
    get
    {
        return _FileName;
    }
    set
    {
        _FileName = value;
    }
}
```

该属性用于设置对话框处理的文本文件名。

在设计器中双击正在设计的窗体的标题栏，则系统打开窗体的 C# 源代码文本，并自动添加窗体加载时的事件处理方法，在方法中输入以下代码：

```
private void frmEdit_Load(object sender, EventArgs e)
{
    if (File.Exists(_FileName))
    {
        using (StreamReader reader = new StreamReader(
            this.FileName,
            Encoding.Default,
            true ))
        {
            txtContent.Text = reader.ReadToEnd();
        }
    }
}
```

```
        _ContentEncoding = reader.CurrentEncoding;
        txtContent.Modified = false;
        this.Text = _FileName;
    }
}
else
{
    txtContent.Enabled = false;
    btnSave.Enabled = false;
    MessageBox.Show("文件 " + _FileName + " 不存在!");
}
}
```

在这段代码中，程序首先检查指定的文件是否存在，若存在则为此文件创建一个类型为 `System.IO.StreamReader` 的对象实例。在创建它的构造函数中，第一个参数为文件名，第二个参数指明采用系统默认的文本编码格式，第三个参数为布尔值，用于指明自动检测文本文件采用的编码格式。

在这里定义了一个名为 “_ContentEncoding” 的文本编码对象，指明了文件加载时使用的文本编码格式。当保存文本文件时将使用这个编码格式，保持文件的编码格式保持不变。

这段代码中使用了 `System.Text.Encoding` 和 `System.IO.StreamReader` 类型，下面进行详细的说明。

1. `System.Text.Encoding` 类型

`System.Text.Encoding` 类型用于进行文本编码格式处理。纯文本文档保存在文件或流中是需要使用某种编码格式的。常见的编码格式有 ASCII、GB2312、UTF-8、UNICODE 等。相同的字符使用不同的编码格式保存，其字节序列有可能不同。

比如对于文本 “ABC 袁永福”，当使用 Windows 记事本以不同的编码格式保存为文本文件时，其文本文件的二进制内容的十六进制编码如表 14-11 所示。

表 14-11 十六进制编码

编 码 格 式	编 码
GB2312	41,42,43,D4,AC,D3,C0,B8,A3
UTF-8	EF,BB,BF,41,42,43,E8,A2,81,E6,B0,B8,E7,A6,8F
UNICODE	FE,FE,41,00,42,00,43,00,81,88,38,6C,8F,79

一些文本文件（但不是所有的）会在文件开头放上几字节来表明其文本编码格式，例如对于 UTF-8 格式，文件标记头为 “EF,BB,BF”，而 UNICODE 为 “FE,FE”。

在 .NET 程序中，所有的文本在内存中的存储格式统一采用 UNICODE 编码格式。而以纯文本格式存储文件时，比如 XML、HTML 文件等也都会涉及编码格式，如果指定了错误的编码格式，则读取时会出现中文乱码。

很多文本编码格式已经成为国际或国家标准，中国的开发者经常遇到的编码格式如表 14-12 所示。

表 14-12 编码格式

编 码 格 式	代 码 页	说 明
ASCII	20127	表示编码从 0 到 127 的字符，一个字符占用 1 字节
UNICODE	1200	编码 UNICODE 字符，基本上一个字符占用 2 字节
UTF-8	65001	以压缩方式编码 UNICODE 字符
GB2312	936	效果上兼容 ASCII，增加了对简体中文字符的支持
BIG5	950	编码繁体中文，俗称大五码

System.Text.Encoding 类型专门用于进行文本编码，能进行二进制和文本之间的转换。由于.NET 程序中文本在内存中的编码格式是 UNICODE，因此实际上就是进行其他编码格式与 UNICODE 编码格式之间的转换。

System.Text.Encoding 类型没有公开的构造函数，开发者只能通过它的一些静态函数来获得对象实例。该类型常用的属性如表 14-13 所示。

表 14-13 常用的属性

名 称	类 型	说 明
ASCII	System.Text.Encoding	静态属性，用于获得 ASCII 编码格式对象
BodyName	String	获得编码格式名称
CodePage	Int	获得编码格式编号
Default	System.Text.Encoding	静态属性，用于获得操作系统默认使用的编码格式对象。 对于简体中文操作系统为 GB2312
EncodingName	String	获得编码格式的名称
Unicode	System.Text.Encoding	静态属性，用于获得 UNICODE 编码格式对象
UTF-7	System.Text.Encoding	静态属性，用于获得 UTF-7 编码格式对象
UTF-8	System.Text.Encoding	静态属性，用于获得 UTF-8 编码格式对象

System.Text.Encoding 类型常用的方法如表 14-14 所示。

表 14-14 常用的方法

签 名	说 明
int GetByteCount (char[])	计算对字符数组所有的字符进行编码所产生的字节数。参数为指定的字符数组
int GetByteCount(string)	计算对指定字符串中所有的字符进行编码所产生的字节数
int GetByteCount (char[] chars , int index , int count)	计算对指定字符数组中的一部分进行编码所产生的字节数。第一个参数为字符数组，第二个参数为参与计算的开始序号，第三个参数为要计算部分的长度
byte[] GetBytes(char[])	对指定字符数据中所有的字符进行编码并返回编码所得的字节数组
int GetCharCount(byte[])	计算对指定字节数组内所有的字节进行解码所产生的字符个数
int GetCharCount(byte[] , int , int)	计算对指定字节数组的指定部分的字节进行解码所产生的字符个数。方法第一个参数为指定的数组，第二个参数为指定部分的开始序号，第三个参数为数组指定部分的长度
char[] GetChars(byte[])	将指定字节数组内所有的字节进行解码生成一组字符

续表

签 名	说 明
char[] GetChars(byte[] , int , int)	将指定字节数组内的指定部分字节进行解码生成一组字符
Encoding GetEncoding(int)	获得指定代码页的编码对象的静态方法。参数为代码页号，若为 0 则返回系统默认的编码。例如使用“Encoding.GetEncoding(936)”就能获得 GB2312 的文本编码对象
Encoding GetEncoding(string)	获得指定名称的编码对象的静态方法。参数为编码名称。例如使用“Encoding.GetEncoding(“gb2312”)”就能获得 GB2312 的文本编码对象
Encoding[] GetEncodings()	获得所有系统支持的编码对象的静态方法
string GetString(byte[])	对指定字节数组的所有字节进行解码生成一个字符串
string GetString(byte[] , int , int)	对指定字节数组的部分字节进行编码生成一个字符串

2. System.IO.StreamReader 类型

类型 System.IO.StreamReader 就是文本数据读取器，能从一个文件或流中以只读向前的方式读取文本内容，默认采用 UTF-8 编码格式，而不是系统默认的编码格式。

StreamReader 派生自类型 System.IO.TextReader，TextReader 还派生了 System.IO.StringReader 类型。

该类型常用的构造函数如表 14-15 所示。

表 14-15 常用的构造函数

参 数	说 明
System.IO.Stream	在一个数据流的基础上创建一个文本读取器，采用系统默认的文本编码格式，参数为基础数据流对象
string	打开指定文件创建文本读取器，参数为一个文件名
System.IO.Stream ,bool	在一个数据流的基础上创建文本读取器。第一个参数为基础数据流对象，第二个参数指示是否自动判断文本编码格式
System.IO.Stream ,System.Text.Encoding	在一个数据流的基础上以指定的文本编码格式创建文本读取器。第一个参数为基础数据流对象，第二个参数为指定的文本编码格式对象
string , bool	打开指定文件创建文本读取器。第一个参数为文件名，第二参数指示是否自动判断文本编码格式
string ,System.Text.Encoding	打开指定文件创建文本读取器。第一个参数为文件名，第二个参数为指定的文本编码格式对象
System.IO.Stream , System.Text.Encoding, bool	在一个数据流的基础上创建文本读取器。第一个参数为基础数据流对象，第二个参数为指定的文本编码格式，第三个参数用于指示是否自动判断文本编码格式，若没有检测到文件使用的编码格式，则使用指定的编码格式
string ,System.Text.Encoding , bool	打开文件创建文本读取器。第一个参数为文件名，第二个参数为指定的文本编码格式，第三个参数用于指示是否自动判断文本编码格式，若没有检测到文件使用的编码格式，则使用指定的编码格式

StreamReader 常用的属性如表 14-16 所示。

表 14-16 常用的属性

名 称	类 型	说 明
BaseStream	System.IO.Stream	返回对象使用的基础数据流对象
CurrentEncoding	System.Text.Encoding	返回当前使用的文本编码格式
EndOfStream	bool	判断当前流的位置是否到底了

StreamReader 常用的方法如表 14-17 所示。

表 14-17 常用的方法

签 名	说 明
void Close ()	关闭流对象
void Dispose()	关闭对象，释放所有的资源
int Peek()	获得当前位置处的下一个可用字符，但不更新当前流的位置。若读取失败则返回-1
int Read()	读取一个字符，并将当前流的位置移动到下一个字符。若读取失败则返回-1
int Read(char[] buffer ,int index , int count)	读取多个字符，并移动当前流的位置。参数 buffer 为保存读取的字符在数组，index 为缓存数组中开始保存的位置，count 为试图读取的字符个数。方法返回实际读取的字符数，不一定等于 count 值
string ReadLine()	读取一行字符，并移动当前位置。方法返回读取的字符串。若读取失败则返回空引用
string ReadToEnd()	读取流中剩余的所有字符，并更新流的当前位置。方法返回读取的字符串，若当前位置已经到了流的尾部，则返回空字符串

在窗体设计界面中双击“保存”按钮，为该按钮添加以下的点击事件处理代码：

```
using (StreamWriter writer = new StreamWriter(  
    _FileName,  
    false,  
    _ContentEncoding))  
{  
    writer.Write(txtContent.Text);  
    txtContent.Modified = false;  
}
```

在这段代码中，首先打开当前文件名指定的文件，创建一个文本书写器对象，然后调用 Write 方法将文本框中的文本内容写入文件，最后设置文本框控件的 Modified 属性为 false。

文本框控件具有可读写属性 Modified，该属性用于指示文本框的内容是否发生改变。当用户修改了文本框的内容，或者应用程序通过设置文本框的 Text 属性修改了内容时，都会导致文本框控件的 Modified 属性值返回 True，以此可以判断文本框内容是否修改了。此处经过保存操作，设置文本框的 Modified 属性值为 False，标记文本框的内容没有被修改，这样若用户没有修改文本内容，则再次保存文件就没有意义了。

3. System.IO.StreamWriter 类型

System.IO.StreamWriter 类型的作用和 System.IO.StreamReader 相反，能以指定的文本编码格式向文件或数据流输出文本数据，它默认采用 UTF-8 编码格式。

System.IO.StreamWriter 类型派生自 System.IO.TextWriter 类型，TextWriter 类型还派生了

System.IO.StringWriter、System.Web.HttpWriter、System.Web.UI.HtmlTextWriter 类型。

StreamWriter 类型常用的构造函数如表 14-18 所示。

表 14-18 常用的构造函数

参 数	说 明
System.IO.Stream	根据一个基础数据流使用默认文本编码格式创建文本书写器
string	打开一个文件，使用默认的文本编码格式创建文本书写器。参数为文件名
System.IO.Stream, System.Text.Encoding	根据一个数据流，使用指定的编码格式创建文本书写器
string, bool	打开一个文件，创建文本书写器。第一个参数为文件名，第二个参数用于指明是否采用追加内容的方式。若该参数为 false，则会删除文件中已有的内容，重新写内容；若为 true 则保留文件中已有的内容，接着向文件中写新的内容
string, bool, System.Text.Encoding	打开一个文件，创建文本书写器。第一个参数为文件名，第二个参数用于指定是否采用追加内容的方式，第三个参数为指定的文本编码格式

StreamWriter 常用的属性如表 14-19 所示。

表 14-19 常用的属性

名 称	类 型	说 明
AutoFlush	bool	指示是否每次调用 Write 方法以后，将其缓冲区中的数据写入到基础流中
BaseStream	System.IO.Stream	用于输出二进制数据的基础流对象
Encoding	System.IO.Encoding	对象使用的文本编码格式

StreamWriter 类型常用的方法如表 14-20 所示。

表 14-20 常用的方法

签 名	说 明
void Close()	关闭对象以及基础流
void Flush()	清理当前编写器的所有缓冲区，并将所有缓冲数据写入到基础流中。StreamWriter 内部会维持一个缓存区，调用对象的 Write 方法写入的数据有可能保存在缓存区中
void Write(paramter)	将一个对象的文本写入到基础流中。参数可以为各种类型，方法内部会将参数对象转换为字符串
void Write(char[])	写入多个字符
void Write(char[] chars, int index, int count)	写入指定字符数组中的指定部分内容。第一个参数为字符数组，第二个参数为要写入部分的开始序号，第三个参数为要写入的部分的字符个数
void WriteLine(parameter)	与 Write 方法功能类似，参数也一样，但在写入文本数据后紧接着写换行符号

新增的这行代码的作用是设置窗体的 DialogResult 属性值。使用以下代码为这个窗体定义一个公开属性：

```
public string TextContent
{
    get
    {
        return txtContent.Text;
    }
}
```

```
    }  
    set  
    {  
        txtContent.Text = value;  
    }  
}
```

该属性能获得或设置窗体文本框中的文本内容。

双击窗体中的“确定”按钮，系统显示了该按钮的点击事件处理 C# 代码，填写以下代码：

```
this.DialogResult = System.Windows.Forms.DialogResult.OK;  
this.Close();
```

这段按钮点击事件处理代码的作用是设置窗体的对话框状态，然后调用 `Close` 方法来关闭窗口体。

到此已实现查看和编辑文本内容的功能。下面需要在主界面中调用这个功能。

在窗体设计器中双击“关闭”按钮，在显示出的 C# 代码文本中输入该按钮的点击事件处理代码：

```
this.Close();
```

这样，当用户单击“关闭”按钮时就能关闭对话框了。

为该窗体添加以下 C# 代码：

```
protected override void OnClosing(CancelEventArgs e)  
{  
    if (txtContent.Modified)  
    {  
        DialogResult result = MessageBox.Show(  
            this,  
            "文件已修改，是否保存？",  
            "系统提示",  
            MessageBoxButtons.YesNoCancel,  
            MessageBoxIcon.Question);  
        if (result == System.Windows.Forms.DialogResult.Yes)  
        {  
            btnSave_Click(null, null);  
        }  
        else if (result == System.Windows.Forms.DialogResult.No)  
        {  
            //用户单击“否”，不保存，继续执行  
        }  
        else if (result == System.Windows.Forms.DialogResult.Cancel)  
        {  
            //用户单击“取消”，取消关闭对话框  
            e.Cancel = true;  
        }  
    }  
    base.OnClosing(e);  
}
```

这段代码重写了窗体的 `OnClosing` 方法，这段代码将在窗体即将关闭前进行一些操作。方法的参数为 `System.ComponentModel.CancelEventArgs` 类型，该类型有一个 `Cancel` 属性，当设置该属性值为 `True` 时，能取消窗体关闭事件，让窗体继续显示。基础对象的 `OnClosing` 方法能触发

窗体的 Closing 事件。

窗体还有一个 OnClosed 方法或 Closed 事件，开发者可以为此编写一些代码进行一些操作，但时间太晚了，不能挽回窗体注定要被关闭的结局。而在早些执行 OnClosing 方法或 Closing 事件是能挽回窗体被关闭的结局的。因此 OnClosing 方法和 OnClosed 方法是有区别的。

在这段重写的 OnClosing 方法中，系统检测文本框内容是否改变，若改变了则显示一个如图 14-9 所示的对话框，提示用户进行保存。

在该对话框中，若用户选择“是”，则系统保存文件，然后继续执行，窗体将关闭；若用户选择“否”，则不进行任何功能操作，继续执行，窗体将关闭；若用户选择“取消”则设置事件参数的 Cancel 属性值为 true，将取消窗体关闭操作，窗体继续显示在用户界面中。

最后调用“base.OnClosing 方法”，一般情况下，在重写控件的方法中还需要调用基础类型的被重写的方法。因为基础类型的方法会触发一些事件，这些事件能运行一些开发者编写的其他代码。

这样查看和编辑文本内容的对话框开发完毕，该对话框运行时的用户界面如图 14-10 所示。

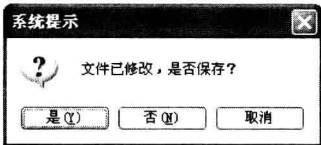


图 14-9 “系统提示”对话框

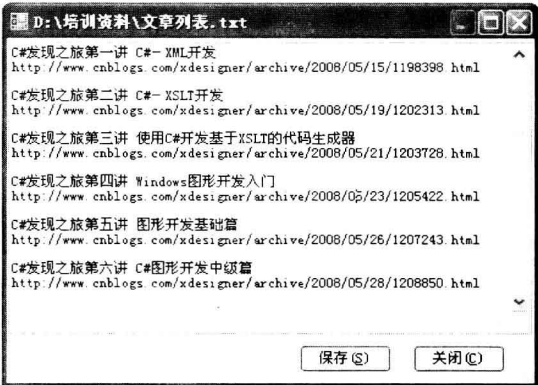


图 14-10 用户界面

14.2.5 查看图片内容

本程序还需要实现能查看图片内容的对话框。为此新建一个名为 frmImage 的窗体，设置窗体属性值如表 14-21 所示。

表 14-21 frmImage 窗体的属性值

属 性	属 性 值
MinimizeBox	False
StartPosition	CenterScreen
ShowInTaskbar	False
Text	查看图片

在窗体上放置一个 Panel 控件，设置该控件的属性如表 14-22 所示。

表 14-22 Panel 控件的属性

属 性	属 性 值
Name	pnlImage
Anchor	Top, Bottom, Left, Right
AutoScroll	True, 设置本属性的目的是让对象支持自动滚动。当控件内容比较大时，控件会自动显示滚动条，让用户通过滚动操作来查看控件的全部内容
BackColor	Window
BorderStyle	Fixed3D

Panel 控件是一个容器控件，其中能放置其他控件，在 pnlImage 控件中添加一个 PictureBox 控件，设置该控件的属性值如表 14-23 所示。

表 14-23 PictureBox 控件的属性值

属 性	属 性 值
Name	picImage
Location	0, 0
SizeMode	AutoSize

在窗体上添加一个 Button 控件，设置其属性值如表 14-24 所示。

表 14-24 Button 控件的属性值

属 性	属 性 值
Name	btnClose
Anchor	Bottom, Right
Text	关闭(&C)

该窗体用户界面如图 14-11 所示。



图 14-11 设计界面

在该设计界面中，双击“关闭”按钮，系统会显示“关闭”按钮的点击事件处理方法，在该方法中输入以下代码：

```
this.Close();
```

输入以下代码定义名为 `FileName` 的属性，该属性用于设置要显示的图片文件名。

```
private string _FileName = null;
public string FileName
{
    get
    {
        return _FileName;
    }
    set
    {
        _FileName = value;
    }
}
```

输入以下代码重写窗体的 `OnLoad` 方法。

```
protected override void OnLoad(EventArgs e)
{
    if (System.IO.File.Exists( this.FileName))
    {
        try
        {
            System.Drawing.Image img
                = System.Drawing.Image.FromFile(this.FileName);
            picImage.Image = img;
            this.Text = this.FileName;
        }
        catch (Exception ext)
        {
            MessageBox.Show(ext.Message);
        }
    }
    base.OnLoad(e);
}
```

在这段代码中，系统调用 `System.Drawing.Image` 类型的静态函数 `FormFile` 来打开指定的文件，并创建一个图片对象，然后设置 `picImage` 控件的 `Image` 属性值。

这样，查看图片内容的对话框就开发完毕了，本对话框运行时的用户界面如图 14-12 所示。

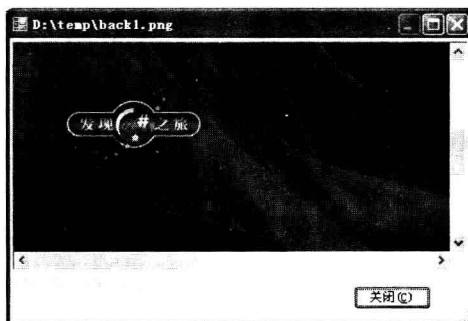


图 14-12 用户界面

14.2.6 访问文件内容

创建了 `frmEdit` 窗体和 `frmImage` 窗体后，程序提供了访问文本文件和图片文件内容的功能。现在需要在本程序段的主窗体中应用这个功能。

打开主窗体 `frmMain` 的设计界面。双击其中的“打开文件”按钮，系统显示了该按钮的点击事件处理方法的代码，在这个方法中输入以下代码：

```

if (lvwFileSystem.SelectedItems.Count > 0
    && lvwFileSystem.SelectedItems[0].Tag is FileInfo)
{
    FileInfo file = (FileInfo)lvwFileSystem.SelectedItems[0].Tag;
    //获得文件扩展名
    string ext = file.Extension;
    if (ext != null)
    {
        ext = ext.Trim().ToLower();
        if (ext == ".txt")
        {
            //使用文本编辑对话框来显示和编辑文本内容
            using (frmEdit dlg = new frmEdit())
            {
                dlg.FileName = file.FullName;
                if (dlg.ShowDialog(this) == DialogResult.OK)
                {
                    //修改文件，刷新列表
                    tvwFileSystem_AfterSelect(null, null);
                }
            }
        }
        else if (ext == ".jpg"
            || ext == ".jpeg"
            || ext == ".bmp"
            || ext == ".gif"
            || ext == ".png")
        {
            //显示图片内容对话框，显示图片文件内容
            using (frmImage dlg = new frmImage())
            {
                dlg.FileName = file.FullName;
                dlg.ShowDialog(this);
            }
        }
    }
} //if

```

在这个方法中，程序获得文件列表中的当前项目，获得文件信息，然后根据文件扩展名创建 `frmEdit` 或 `frmImage` 类型的窗体，设置文件名属性，最后调用 `ShowDialog` 方法来显示模式对话框。

对于 `frmEdit` 类型的对话框，若它的 `ShowDialog` 方法返回 `DialogResult.OK`，说明在对话框内部修改了文件内容，此时文件大小会变化，程序会刷新文件列表来反映最新的状态。

这样，一个简单的文件系统操作应用程序就开发完毕了，该程序的运行界面如图 14-13 所示。

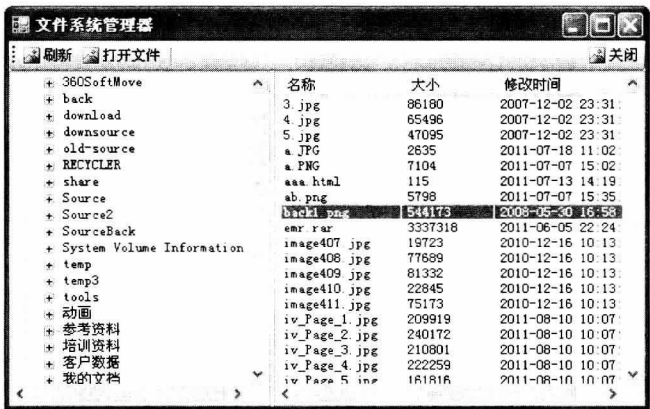


图 14-13 文件系统管理器

在图 14-13 中，左边列出了计算机中所有的磁盘以及文件目录树状结构，右边列出了当前目录下所有的子目录和文件清单。



关系型数据库开发基础

现在的系统大部分都是信息管理系统，都是基于关系型数据库开发的，应用系统大部分功能针对数据库的增、删、改、查。因此关系型数据库成为应用系统中最为重要的底层部分。每一个软件开发者都必须掌握基础的关系型数据库开发知识。本章将介绍这些基础知识。

15.1 主流数据库介绍

关系型数据库发展历史很久了，从理论上讲，它有坚实的数学基础；从工程上看，它有广泛的实践应用，而且有很多成熟的软件产品。

对于开发者来说，关系型数据库就是一个大的容器，其中包含了一些二维数据表格，每个数据表格可以存储多条记录。可以使用标准的 SQL 语句来实现数据库记录的增、删、改、查的功能。具体原理可以参考相关书籍。

目前开发者能使用很多关系型数据库管理系统，下面简单介绍常用的数据库管理系统。

15.1.1 MS Access

MS Access 数据库是微软提供的数据库产品，是微软 Office 办公套件中的一部分，是应用最为广泛的通用数据库管理系统。

MS Access 数据库是一种存储在本地文件系统中的单文件数据库系统，数据库文件名以 mdb 为扩展名，可以使用微软 Office 办公套件中的 Access 软件来设计和修改 Access 数据库。Access 数据库有多种版本，目前使用比较多的是 Access 2000。

图 15-1 是 MS Office 2003 中的 Access 软件打开 Access 数据库时的用户界面。

1. 特点

Access 数据库的特点主要有：

(1) 使用简单，可利用强大的 MS Office Access 软件编辑数据库文件，能与 Excel 协同工作。

(2) 部署方便，它是单文件数据库系统，一个数据库文件能放置所有的内容，因此部署非常方便。进行文件复制操作就能实现完整的数据备份。



图 15-1 Microsoft Access

(3) 开发简单，所有的 Windows 系统都具有访问 Access 数据库的 JET 数据库引擎，能通过 OLEDB、ADO.NET 等技术访问 Access 数据库，软件开发比较简单。

(4) 具有一定的存储能力，数据库文件大小可以为上百 MB，能存放几十万条记录。

(5) 具有内置开发能力，能存储表单、VBA 代码等，很多非专业的 IT 人士都使用它的这种能力开发一些小型应用。

由于具有以上特点，因此 Access 数据库被广泛应用于很多中小型信息系统，还被许多系统作为交换和导出数据的存储格式，且在开发过程中经常作为原型系统使用的数据库。

2. 缺点

Access 数据库属于轻量级数据库，其缺点有：

(1) 对于大型和复杂的系统，其功能还不够强大，当存储上百万条记录时性能比较差，而且不支持触发器及存储过程。

(2) Access 是基于本地文件系统的，不能很方便地通过网络访问数据库文件，没有多少用户权限控制。

由于 Access 数据库能力有限，因此 Access 数据库一般是不用做大中型信息系统开发的。不过在业界小型应用系统中用的还是非常多的，而且它也很适合非专业 IT 人士使用，因此 Access 数据库应用非常广泛。

3. 常用操作

人们一般都使用 MS Office 中的 Access 程序对 Access 数据库进行操作，在此使用 Access 2003 进行演示，常用的操作有以下几种。

(1) 新建数据库

在如图 15-2 所示的界面中，单击主菜单中的“文件→新建”，则右边会显示“新建文件”任务栏，单击“空数据库”，系统会弹出一个数据库文件名选择对话框，让用户输入数据库文件名。用户确认操作后程序会创建指定名称的 Access 2000 格式的数据库文件并打开它，此时程序用户界面如图 15-3 所示。

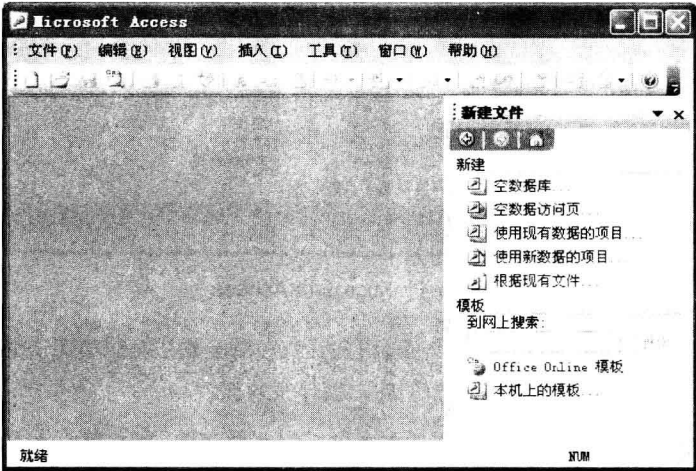


图 15-2 “新建文件”任务栏

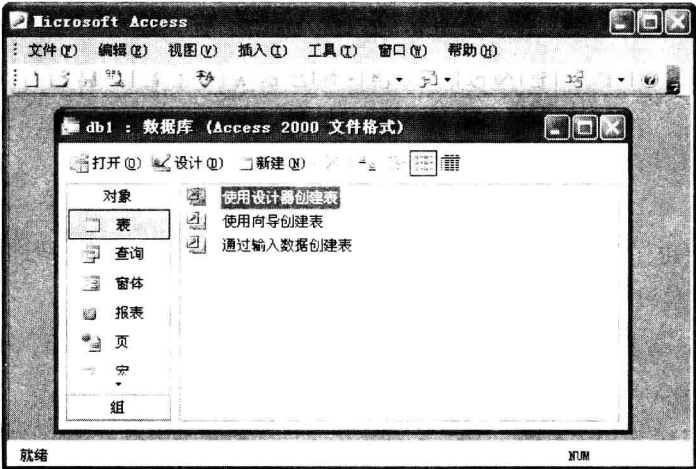


图 15-3 用户界面

(2) 新建数据表

打开数据库文件后，可以双击用户界面中的“使用设计器创建表”来显示如图 15-4 所示的数据表设计界面。

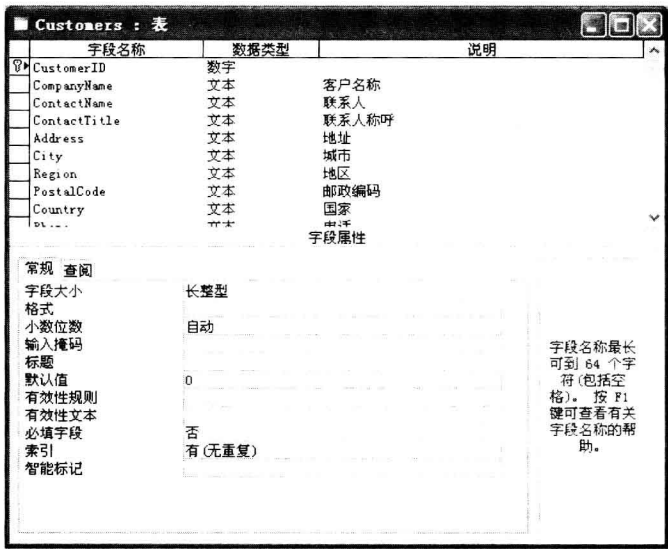


图 15-4 数据表设计界面

在该界面中，开发者可以在上面的列表中输入字段名称、数据类型和说明，在下面的属性列表中设置字段的各种属性。

Access 2000 中常用的字段数据类型如表 15-1 所示。

表 15-1 Access 2000 中常用的字段数据类型

字段数据类型	说 明
文本	变长度的文本，最多 255 个字符
备注	大文本，长度不限
数字	数字字段，可以在下面的属性列表中进一步地设置为整数或浮点数
日期/时间	日期/时间类型的数据
货币	货币数值类型的数据，类似于 C# 中的 decimal 数据类型
自动编号	能自动编号的整数。新增记录时会自动计算分配字段值
是/否	相当于 C# 中的 bool 数据类型
OLE 对象	大字段，能存放二进制数据，长度不限，相当于 C# 中的字节数组

对于某个字段用鼠标右击可弹出如图 15-5 所示的快捷菜单。单击其中的“主键”菜单项目，就可以设置某个字段是否是关键字段。


设计完数据表的字段后，单击工具条上的“保存”按钮，则程序会弹出如图 15-6 所示的对话框。



图 15-5 快捷菜单

在该对话框中输入新增的数据表的名称，单击“确定”按钮，此时数据库的数据表的列表中就出现了刚刚新建的数据表。

(3) 修改数据表

在数据表的列表选中某个数据表，单击工具条上的“设计(D)”按钮，则会显示出该数据表的字段设计器，在该设计器中可以修改数据表的字段。若数据表中已经有一些数据，则保存设计的时候系统会进行一些提示。

在数据表的列表中，用鼠标右击可弹出快捷菜单，单击“属性”菜单项目会弹出如图 15-7 所示的数据表属性对话框。



图 15-6 “另存为”对话框

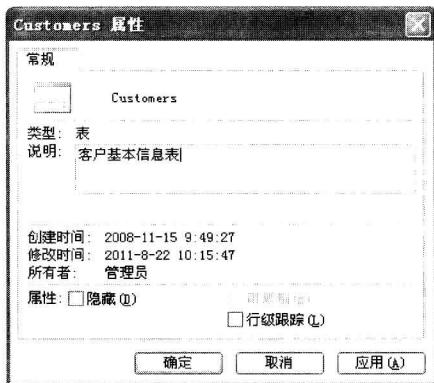


图 15-7 属性对话框

在该对话框中，用户可以设置关于数据表的一些说明，一般为数据表的中文名。当为数据表设置了说明后，数据表的列表如图 15-8 所示。此时数据库更容易理解和使用。

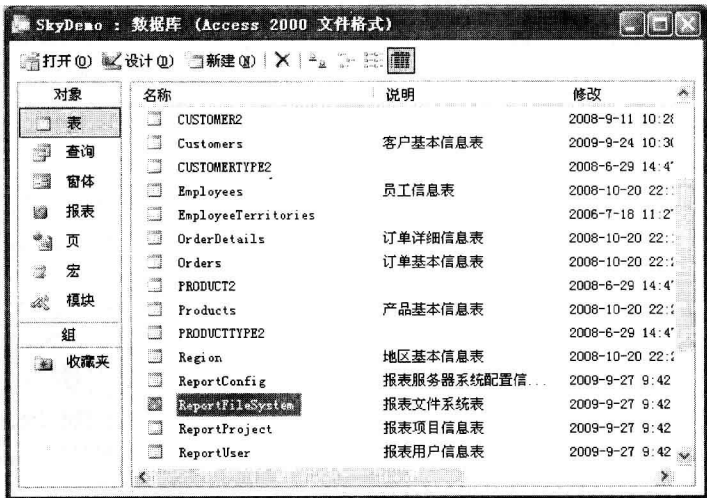


图 15-8 数据表列表

(4) 删除数据表

在数据表的列表中选中某个数据表，单击菜单项目“编辑→删除”，则系统会弹出如图 15-9 所示的对话框。

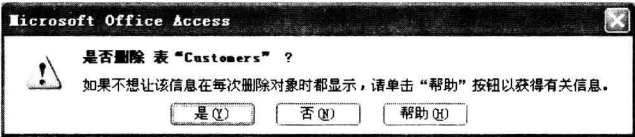


图 15-9 确认删除表对话框

用户确认操作后程序就会删除数据表，删除的数据表不可恢复。

(5) 查看和修改表数据

在数据表的列表中双击某个数据表对象时，能显示如图 15-10 所示的数据表数据查看和修改界面。

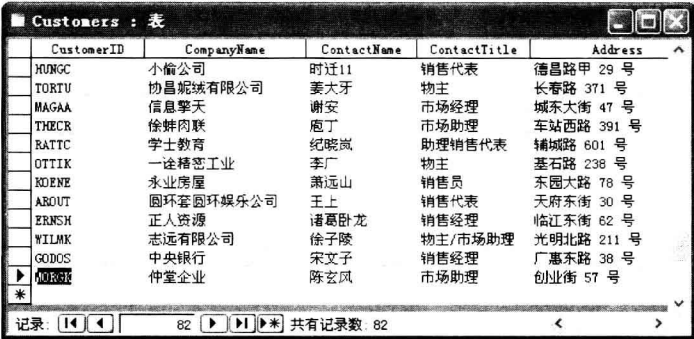


图 15-10 数据查看和修改界面

在该界面中，用户可以查看和输入记录，可以在标记了“*”图标的那一行输入新增记录的内容。

用户也可以在表格行头区域通过鼠标拖曳操作选中多条记录，然后按下 Delete 键进行删除记录的操作，在删除前系统会显示如图 15-11 所示的对话框来让用户确认删除操作。

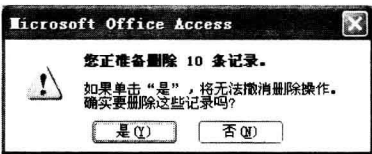


图 15-11 确认删除操作对话框

由于 Access 程序是属于 MS Office 成员的，因此在这个数据编辑界面中复制的数据可以粘贴到 MS Word 或 MS Excel 中，而 MS Word 或 MS Excel 中复制的表格数据也可以粘贴到

Access 中。

(6) 导入数据

单击主菜单中的“文件→获取外部数据→导入”，弹出如图 15-12 所示的数据源文件“导入”对话框。在该对话框中，用户可以展开文件类型下拉列表来选择更多的文件类型。

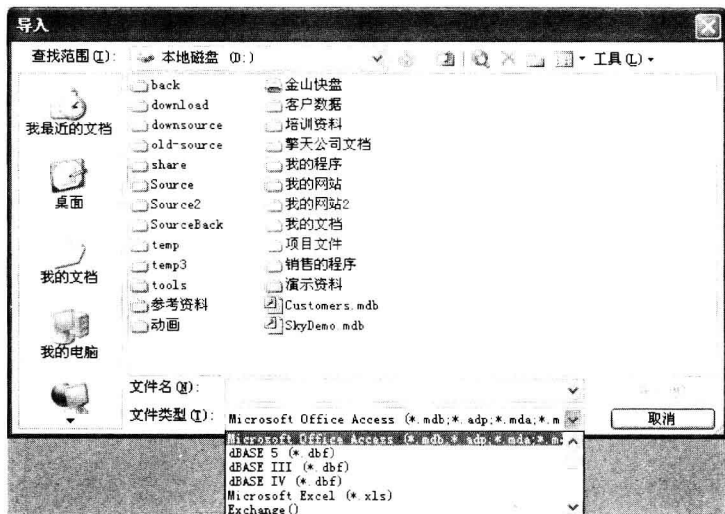


图 15-12 “导入”对话框

若用户选择了其他的 Access 数据库文件，则程序会分析用户选中的 Access 数据库的结构，显示如图 15-13 所示的对话框。

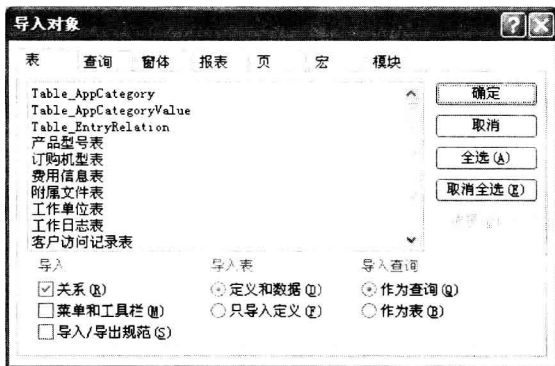


图 15-13 “导入对象”对话框

在该对话框中，用户可以选择要导入的对象，进行一些设置，单击“确定”按钮后就能将一些其他 Access 数据库中的对象导入到当前操作的数据库中。

若用户选中了一个 Excel 工作表文件（扩展名为 xls），则程序会显示如图 15-14 所示的向导对话框。借助于该对话框，用户就能将 Excel 工作表中的数据导入到 Access 数据库中。



图 15-14 向导对话框

(7) 导出数据

在 Access 2003 中，一次导出操作只能导出一张数据表中的数据。用户必须在数据表的列表选中要导出的数据表对象，然后单击主菜单中的“文件→导出”菜单项，程序会显示出“导出”对话框，让用户选择要导出的文件名，在该对话框中可以选中多种格式的数据文件名。

若用户选择了其他的 Access 数据库文件，则系统显示如图 15-15 所示的对话框。单击“确定”按钮即可将数据表导出到其他数据库中。

若用户选择了 Excel 工作表文件，则系统会在用户指定的 Excel 工作表中新建一个工作簿，然后将数据复制到这个工作簿中。

(8) 设置密码

单击主菜单中的“工具→安全→设置数据库密码”菜单项，系统会弹出如图 15-16 所示的对话框。

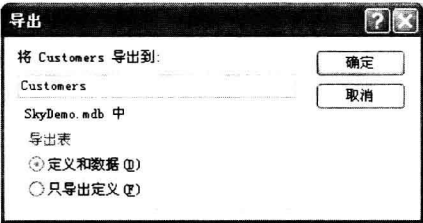


图 15-15 “导出”对话框

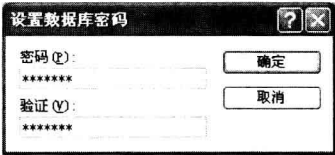


图 15-16 “设置数据库密码”对话框

在该对话框中，用户输入两遍密码，单击“确定”按钮，该数据库就设置了密码。在准备使用相关方式打开该数据库时，比如使用 Office Access 程序或编程连接数据库，都需要指定正确的密码，否则打不开该数据库。

要进行设置密码的操作，需要 Office Access 程序以独占方式打开数据库文件。而在一般情况

下，用户都是在 Windows 资源管理器中双击 Access 文件来打开数据库的，此时并不是以独占方式打开的。若要以独占方式打开，则需运行 Office Access 程序，单击主菜单项目“文件→打开”，程序会显示如图 15-17 所示的对话框。

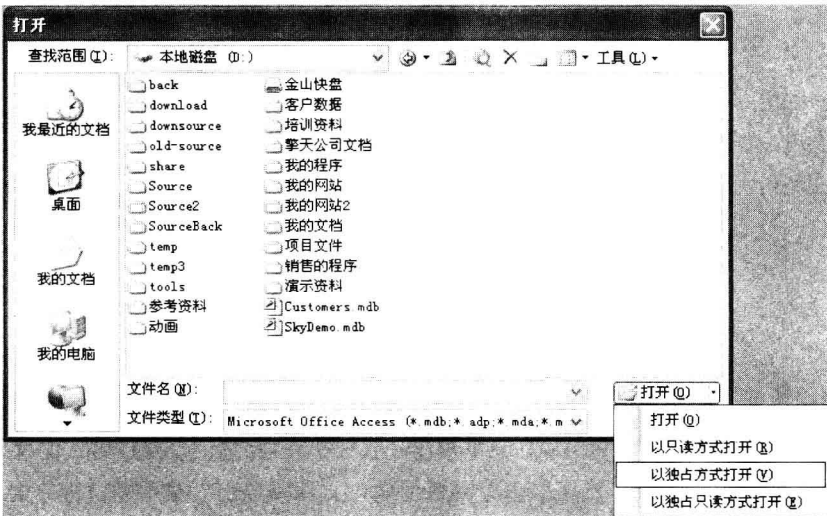


图 15-17 “打开”对话框

在该对话框中，选中要打开的数据库文件，然后单击“打开”按钮右边的下拉箭头，在弹出的菜单中单击“以独占方式打开”，则 Office Access 程序就以独占方式打开数据库文件。

当成功地打开一个设置了密码的数据库后，用户单击主菜单中的“工具→安全→撤销数据库密码”菜单项，此时程序会显示出如图 5-18 所示的对话框，让用户输入当前密码。若用户输入的密码正确，则程序会撤销数据库密码。

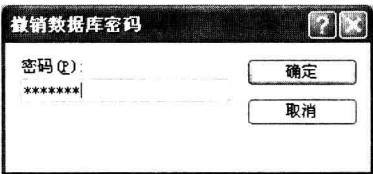


图 15-18 “撤销数据库密码”对话框

(9) 压缩和修复数据库

Access 数据库没有强大的安全机制，因此某些操作会导致数据错误甚至无法打开；而且 Access 没有很好的内容空间收集机制，随着大量记录的增、删、改操作，数据库文件会逐渐变大，内部存在大量的零碎空间而无法利用。

用户单击 Access 程序的主菜单项目“工具→数据库实用工具→压缩和修复数据库”，Access 程序会试图修复数据库文件中的一些错误，并整理数据库文件的内部数据组织结构，清理碎片空间，减少数据库文件。

(10) 数据库排序次序

在中文操作系统中使用 Office Access 程序创建的 Access 数据库文件，被用在英文操作系统中，有可能出现“Selected Collating Sequence Not Supported（选择该操作系统不支持的排序序列）”错误。这是由于中文 Access 为了改善用户体验，采用了适合中文的排序方式，但这种排序

方式是不受英文操作系统支持的。

为了解决这个问题，用户可以在 Office Access 程序中单击主菜单项目“工具→选项”，程序会显示如图 15-19 所示的“选项”对话框。

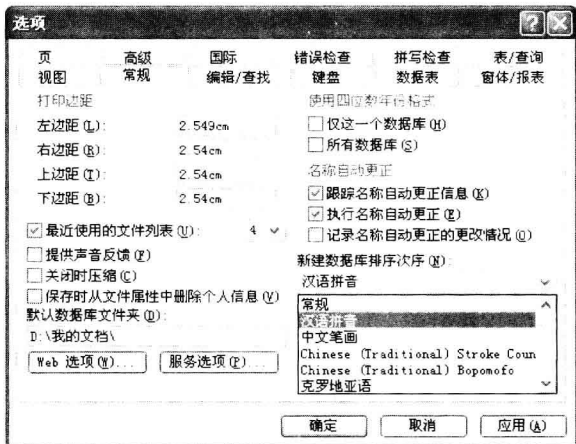


图 15-19 “选项”对话框

在该对话框中选择“常规”选项卡，其中有一个“新建数据库排序次序”下拉列表，当 Access 数据库用于英文操作系统时，在新建数据库前需要在该下拉列表中选择“常规”。

注意，这个选项能让 Office Access 程序今后创建的数据库都来使用指定的排序次序，而对于现有的数据库没有影响。因此对于现有的数据库，只能新建数据库，然后再导入现有的数据库中的全部内容，这样才能解决一些问题。

15.1.2 MS SQL Server

MS SQL Server 是一个比较强大的关系型数据库，最初是由 Microsoft、Sybase、Ashton-Tate 三家公司共同开发的，后来 Microsoft 和 Sybase 在 SQL Server 的开发上分道扬镳，于是现在业界就有了 MS SQL Server 和 Sybase SQL Server 之分。由于 MS SQL Server 远比 Sybase SQL Server 应用广泛，因此一般来说，SQL Server 就是指 MS SQL Server。

MS SQL Server 是一种 C/S 架构的关系型数据库，远比 MS Access 强大，支持多用户、并发、存储过程、触发器等，能容纳海量的数据。

MS SQL Server 有多个版本，常用的有 SQL Server 2000、MSDE 2000、SQL Server 2005。

SQL Server 2000 是使用比较广泛的 SQL Server 版本，它的服务器程序以 Service 的方式运行在 Windows NT/XP 或更高版本的 Windows 操作系统中，其服务的名称为“MSSQLSERVER”，其客户端的管理软件常用的有企业管理器、查询分析器和事件探查器。

1. 企业管理器

单击 Windows 开始菜单项目“所有程序→Microsoft SQL Server→企业管理器”即可运行

SQL Server 企业管理器。企业管理器是一个针对 SQL Server 2000 的功能强大的客户端管理和控制软件，其主界面如图 15-20 所示。

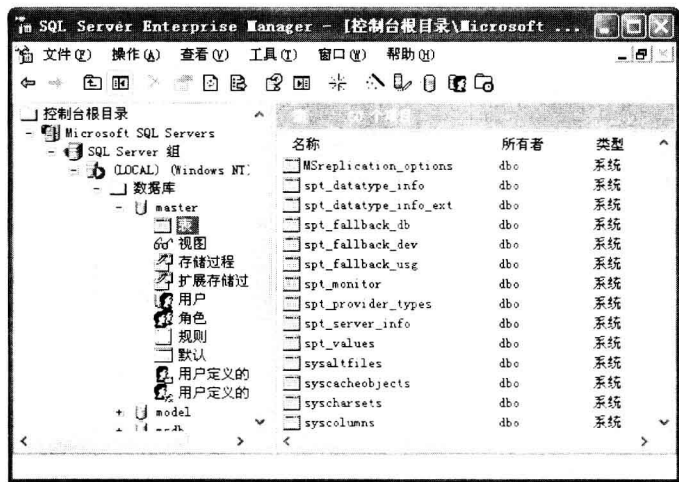


图 15-20 SQL Server 企业管理器主界面

在该主界面中，左边为一个服务器资源树状列表，右边为当前节点下的子项目的视图列表。可以看出一个 SQL Server 服务器可以包含多个数据库，而多个数据库又包含多个数据表。

使用企业管理器，可以完成以下操作。

(1) 连接数据库服务器

在图 15-20 左边的树状列表中选中“SQL Server 组”节点，单击主菜单项目“操作→新建 SQL Server 注册”，程序会显示如图 15-21 所示的对话框。

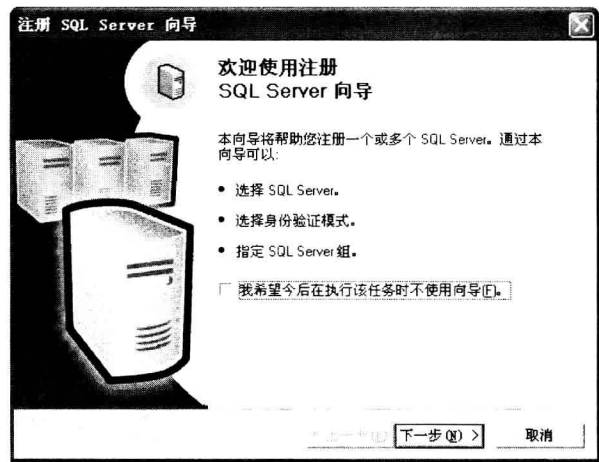


图 15-21 向导对话框

在该对话框中，单击“下一步”按钮，弹出如图 15-22 所示的对话框。

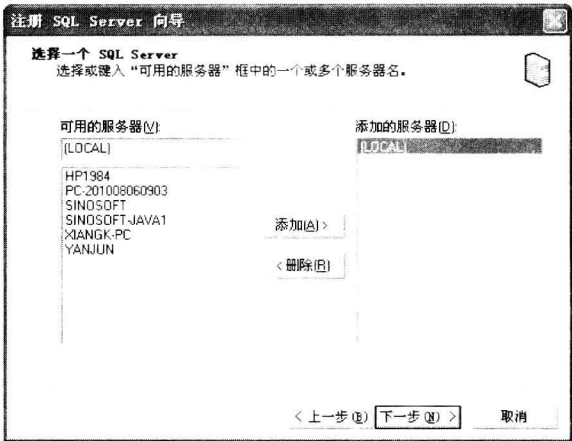


图 15-22 “选择一个 SQL Server” 对话框

在图 15-22 中，左边列出了局域网中搜索到的可用的服务器，用户可以单击列表中的服务器，也可以直接输入服务器的名称或 IP 地址，若输入“(LOCAL)”则表示本机的服务器，然后单击“添加”按钮将其添加到“添加的服务器”列表中。

单击“下一步”按钮，弹出如图 15-23 所示的对话框。

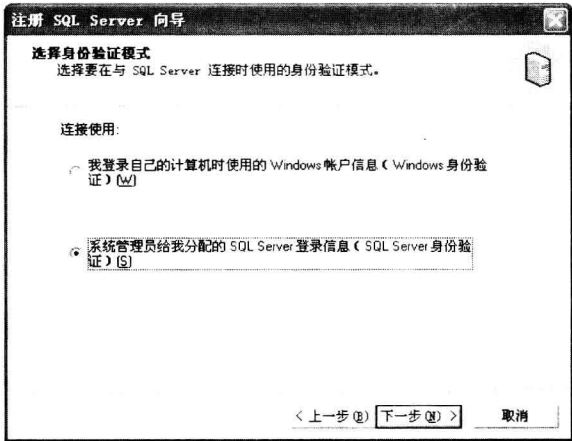


图 15-23 “选择身份验证模式” 对话框

在该对话框中可以设置用户登录的方式，此时需要询问数据库管理员。确定采用的方式后，单击“下一步”按钮，弹出如图 15-24 所示的对话框。

在该对话框中输入登录服务器使用的用户名和密码。然后单击“下一步”按钮，弹出如图 15-25 所示的对话框。

在该对话框中选择要添加到 SQL Server 组中，然后单击“下一步”按钮，弹出如图 15-26 所示的对话框。

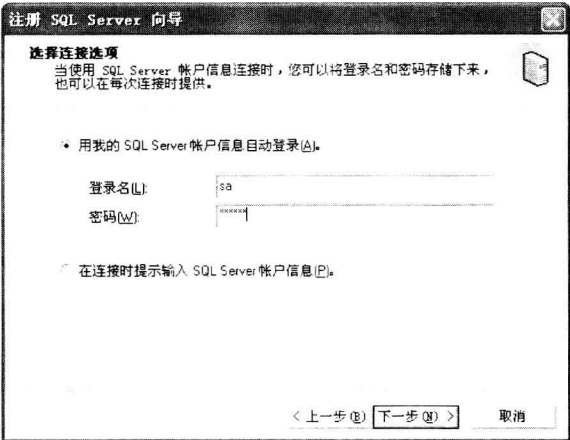


图 15-24 “选择连接选项”对话框

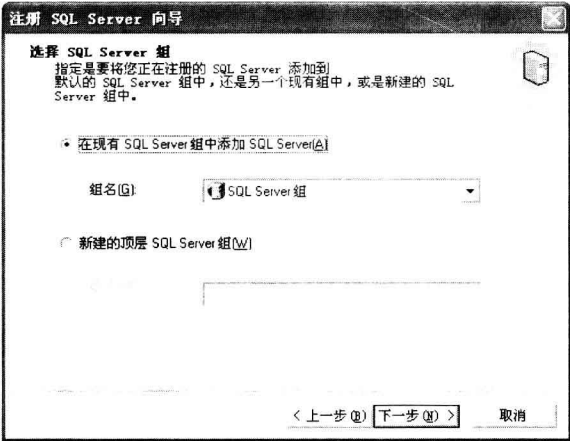


图 15-25 “选择 SQL Server 组”对话框



图 15-26 “完成注册”对话框

此时所有的信息配置完毕，单击“完成”按钮，程序会试图连接数据库服务器。若连接成功，则会显示如图 15-27 所示的提示，并在 SQL Server 企业管理器主界面左边的树状列表中添加数据库服务器节点。

(2) 新建数据库

在 SQL Server 企业管理器主界面左边的树状列表中选中某个服务器节点，用鼠标右击弹出快捷菜单，如图 15-28 所示。

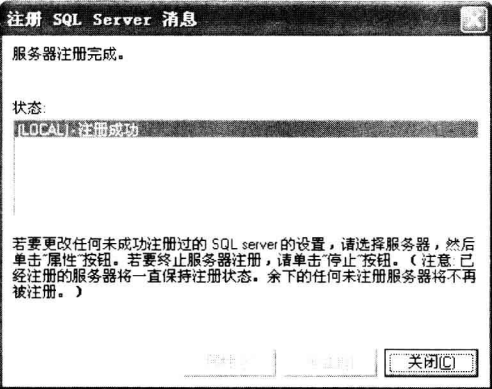


图 15-27 “注册 SQL Server 消息”对话框

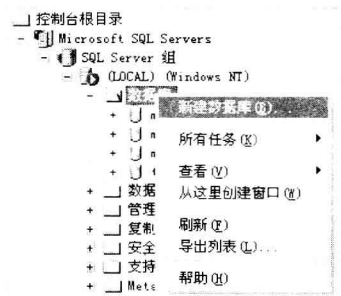


图 15-28 快捷菜单

单击“新建数据库”菜单项目，程序会显示如图 15-29 所示的“数据库属性”对话框。

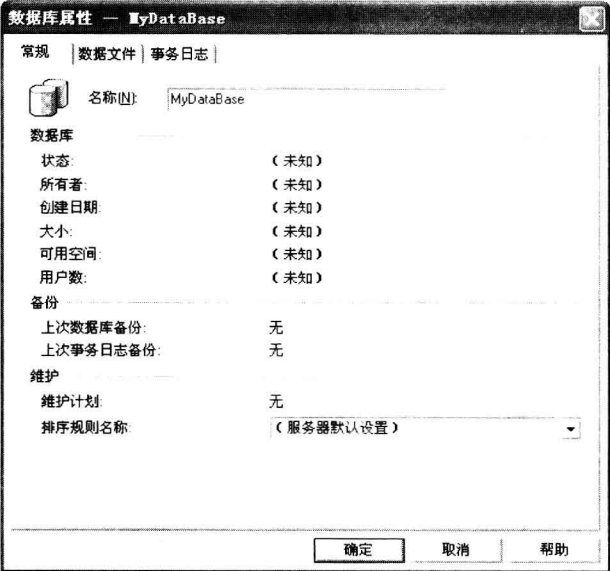


图 15-29 “数据库属性”对话框

在该对话框中输入新建的数据库名称，注意同一个服务器中数据库名称不能重复；然后切换

到“数据文件”选项卡，如图 15-30 所示。

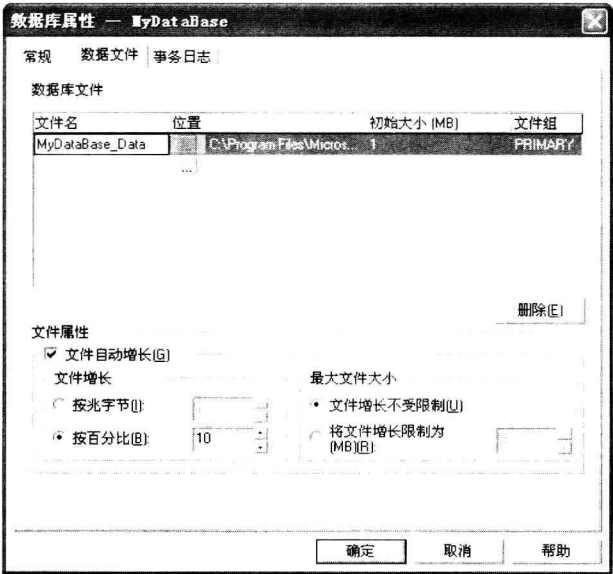


图 15-30 “数据文件”选项卡

在该选项卡下，用户可以设置数据库文件名和一些属性。注意，这里的文件位置是服务器中的文件系统的位置，而不是用户使用的本地计算机中的文件位置。单击“位置”下面的小按钮会显示如图 15-31 所示的对话框。



图 15-31 “查找数据库文件”对话框

通过该对话框就能浏览服务器上的文件目录和文件名。

切换到“事务日志”选项卡，可以设置数据库日志文件名和一些属性，如图 15-32 所示。

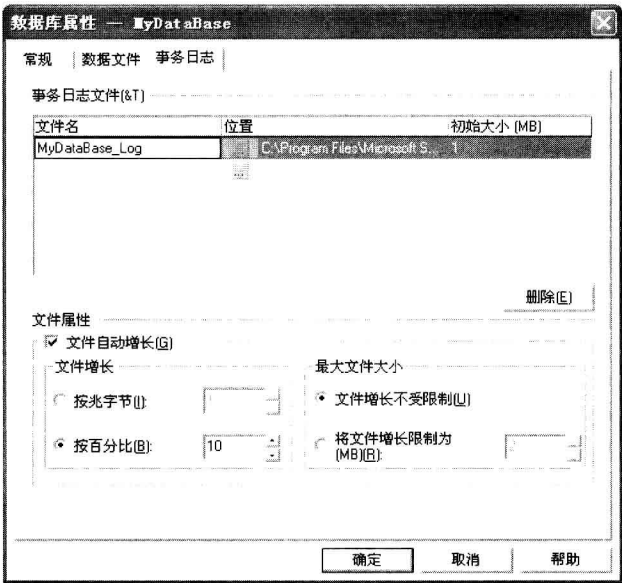


图 15-32 “事务日志”选项卡

单击“确定”按钮，企业管理器就能在 SQL Server 服务器中创建新的数据库。

(3) 备份数据库

在 SQL Server 企业管理器主界面左边的树状列表中选中要备份的数据库，用鼠标右击弹出快捷菜单，单击菜单项目“所有任务→备份数据库”，如图 15-33 所示。

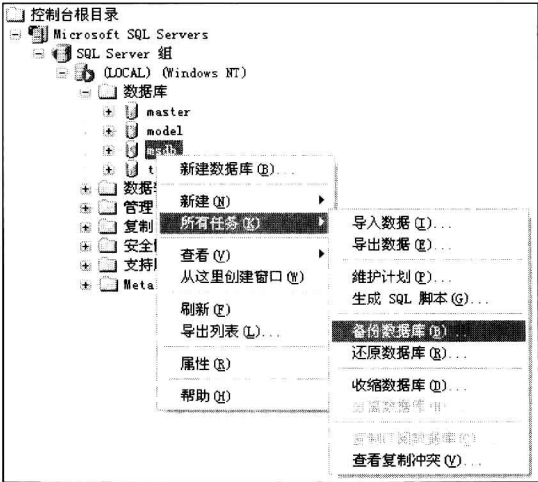


图 15-33 快捷菜单

程序会显示如图 15-34 所示的“SQL Server 备份”对话框。在该对话框中单击“添加”按钮，程序会显示如图 15-35 所示的对话框。

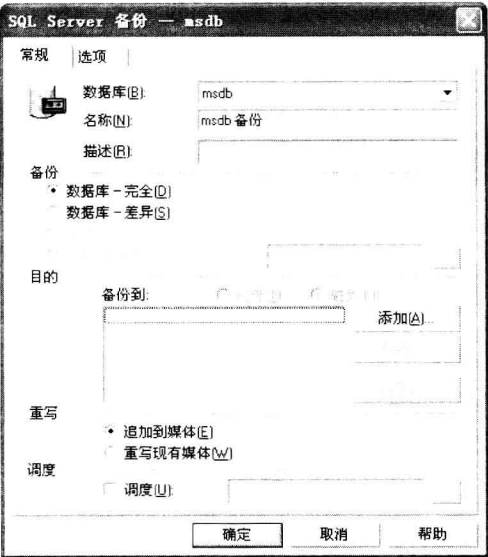


图 15-34 “SQL Server 备份” 对话框

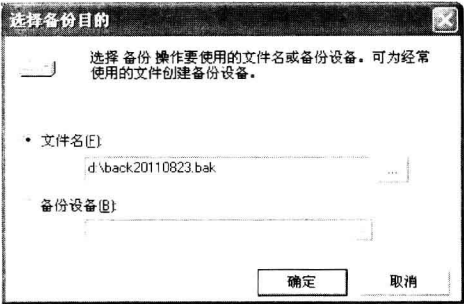


图 15-35 “选择备份目的” 对话框

在该对话框中，用户可以输入或浏览要备份的文件名。注意，这个文件名是服务器中的文件名，而不是用户使用客户端计算机的本地文件名。

单击“确定”按钮关闭对话框，回到“SQL Server 备份”对话框，如图 15-36 所示。

此时在该对话框中，可以看到“目的”列表中出现了用户输入的备份目的文件名。单击“确定”按钮，程序开始进行备份操作，并显示如图 15-37 所示的备份进度消息框。

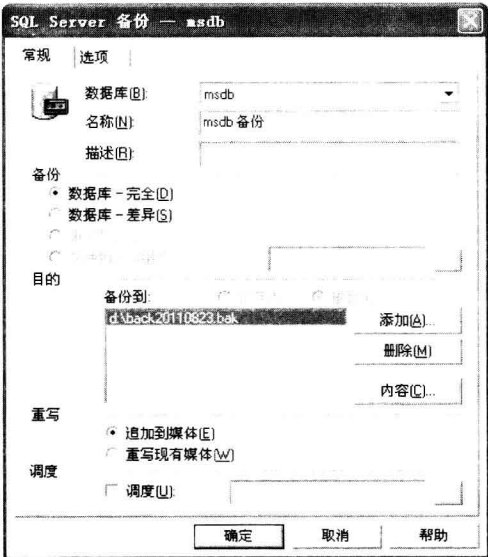


图 15-36 “SQL Server 备份” 对话框

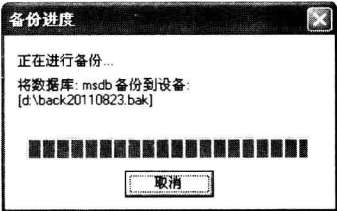


图 15-37 备份进度

若备份操作成功，则程序会提示备份完成。

(4) 还原数据库

在 SQL Server 企业管理器主界面左边的树状列表中选中“数据库”节点，弹出快捷菜单，单击菜单项目“所有任务→还原数据库”，如图 15-38 所示。

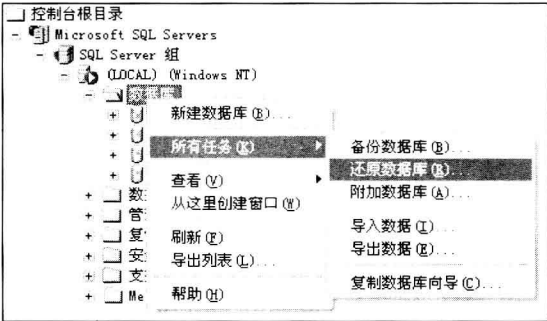


图 15-38 快捷菜单

系统会显示如图 15-39 所示的“还原数据库”对话框。

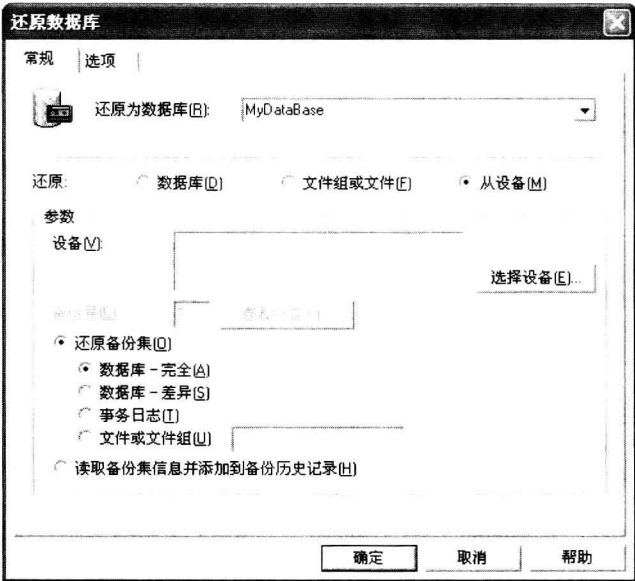


图 15-39 “还原数据库”对话框

在该对话框中输入要还原的数据库名称，然后单击“选择设备”按钮，程序会显示如图 15-40 所示的“选择还原设备”对话框。

在该对话框中单击“添加”按钮，程序会显示如图 15-41 所示的“选择还原目的”对话框。

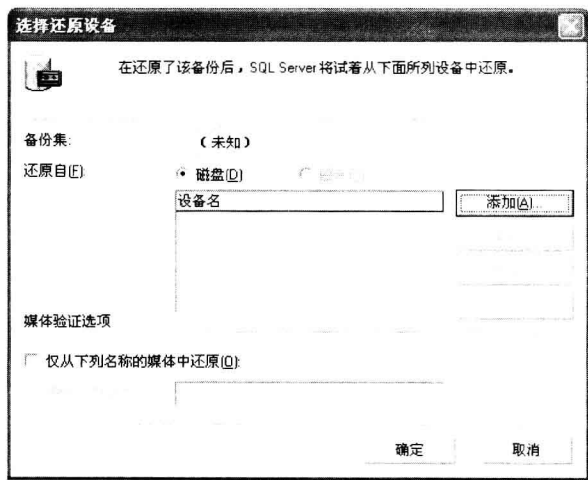


图 15-40 “选择还原设备” 对话框

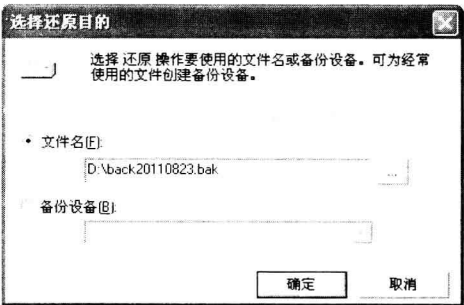


图 15-41 “选择还原目的” 对话框

在该对话框中，用户可以输入文件名，或者单击文件名输入框后面的小按钮浏览选择文件。这里的文件名是服务器上磁盘中的文件名，而不是客户端计算机中的文件名。

用户输入正确的文件名后，单击“确定”按钮，关闭对话框，程序会弹出如图 15-42 所示的“还原数据库”对话框。

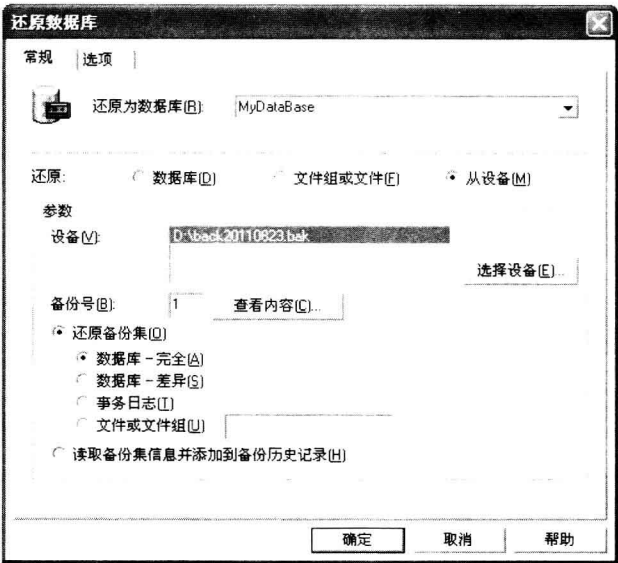


图 15-42 “还原数据库” 对话框

在该对话框中单击“确定”按钮，程序开始进行还原数据库的操作，此时会显示如图 15-43 所示的“还原进度”窗口。

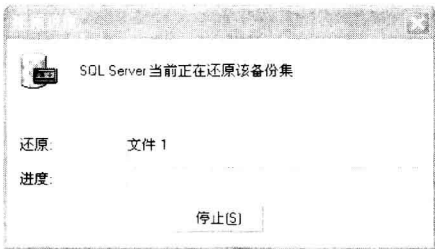


图 15-43 “还原进度”窗口

若还原数据库操作成功，则程序会提示操作完成。

(5) 附加数据库

在实践中还经常采用直接复制数据库文件的方式来备份或迁移数据库，然后使用附加数据库的操作将数据库文件附加到服务器中，实现另外一种数据库的还原操作。

直接复制数据库文件的方式需要停止 SQL Server 服务器，并停止 SQL Server 相关的 Windows 服务，然后将数据库文件（扩展名为 mdf）和配套的日志文件（扩展名为 ldf）复制到另外的存储介质中；而备份数据库操作不需要停止 SQL Server 服务器，可在线执行。

附加数据库的操作过程为，在 SQL Server 企业管理器主界面左边的树状列表中选中“数据库”节点，用鼠标右击弹出快捷菜单，单击菜单项目“所有任务→附加数据库”，程序会显示如图 15-44 所示的对话框。

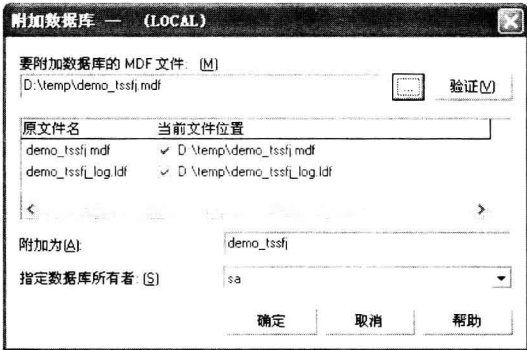


图 15-44 “附加数据库”对话框

在该对话框中，用户输入 SQL Server 数据库文件名（扩展名一般为 mdf），或者单击文件名输入框后面的小按钮浏览选择一个文件名，单击“验证”按钮，会在对话框中间的列表中列出有效的数据库文件名及配套的日志文件名（扩展名为 ldf），单击“确定”按钮，程序会将该数据库文件附加到 SQL Server 服务器中，形成一个新的数据库。

(6) 新建数据表

在 SQL Server 企业管理器主界面左边的树状列表中选中“表”节点，如图 15-45 所示。

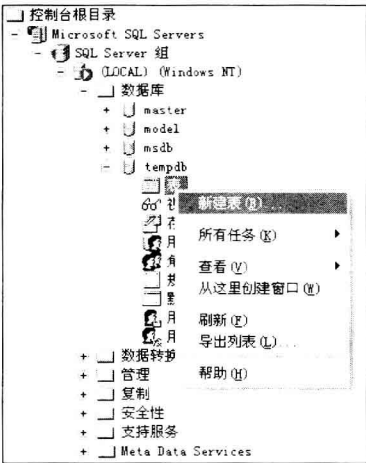


图 15-45 选中“表”节点

用鼠标右击弹出快捷菜单，单击“新建表”菜单项，会显示如图 15-46 所示的数据表设计器。使用该数据表设计器，可以为新的数据表添加字段定义。

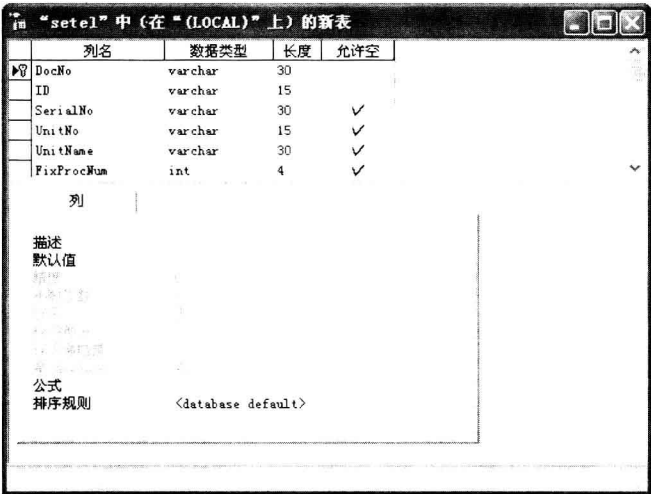


图 15-46 数据表设计器

SQL Server 中常用的字段数据类型如表 15-2 所示。

表 15-2 SQL Server 中常用的字段数据类型

字段数据类型	说 明
binary	固定长度二进制字段，最大可设置为 8000 字节
char	固定长度字符数据，最大可设置为 8000 个字符。若字符内容不足则会在后面补空格以达到指定的长度

续表

字段数据类型	说 明
datetime	日期时间数据
decimal	货币数据
float	单精度浮点数
int	32 位整数
nvarchar	UNICODE 变长字符串，最大可设置为 4000 个字符
text	可变长度的非 UNICODE 文本数据，最多 20 亿个字符
ntext	可变长度的 UNICODE 文本数据，最多 10 亿个字符
image	二进制字段，最多 20 亿字节

数据表字段定义完毕后，单击工具条上的“保存”按钮，程序会弹出如图 15-47 所示的对话框，让用户输入新建的数据表名称。



图 15-47 “选择名称”对话框

在该对话框中，输入数据表名称后，单击“确定”按钮，程序就会在数据库中新建一个数据表。

(7) 修改数据表结构

在 SQL Server 企业管理器主界面左边的树状列表中选中“表”节点，则右边会显示数据库中所有的数据表，选中某个数据表，用鼠标右击弹出快捷菜单，如图 15-48 所示。

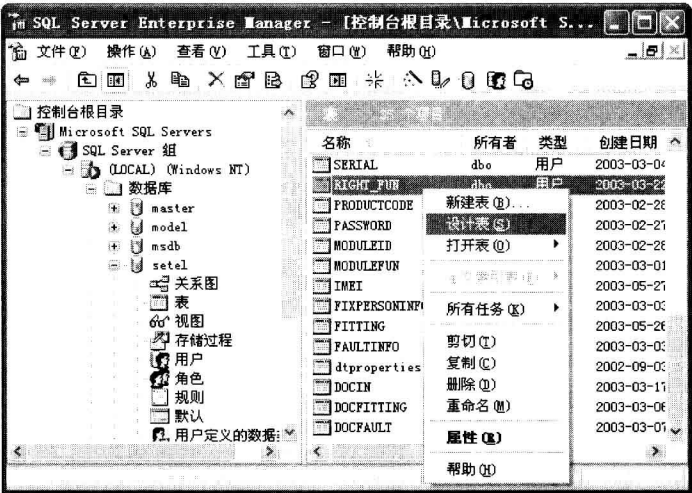


图 15-48 选中某个数据表后的快捷菜单

单击快捷菜单项目“设计表”，程序会显示出该数据表的字段设计器，如图 15-49 所示。

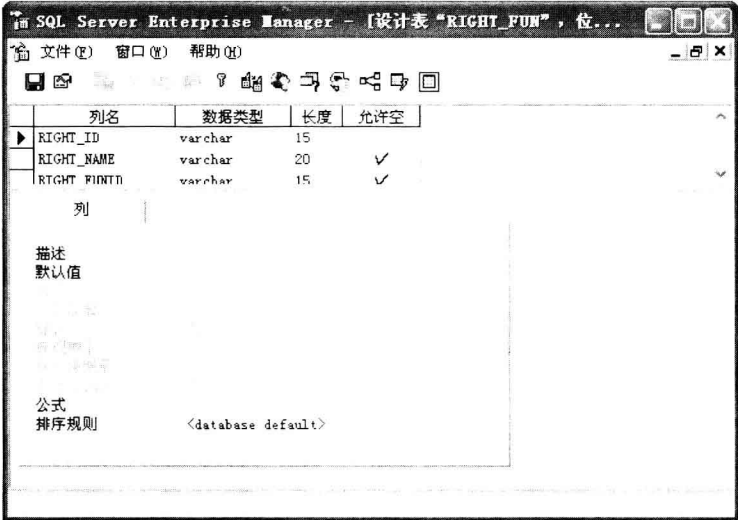


图 15-49 字段设计器

通过该字段设计器，用户就可以查看和修改该数据的字段结构信息。用户修改字段信息后，单击工具条上的“保存”按钮即可修改数据表结构信息。

(8) 查看修改表数据

在 SQL Server 企业管理器主界面右边的数据表列表中选中某个数据表，用鼠标右击弹出快捷菜单，如图 15-50 所示。单击菜单项目“打开表→返回所有行”，则程序查询该数据表中所有的记录并显示出来，如图 15-51 所示。

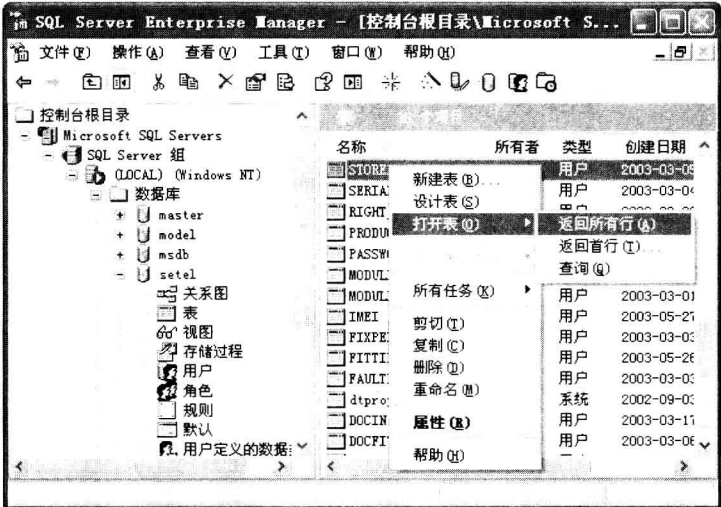
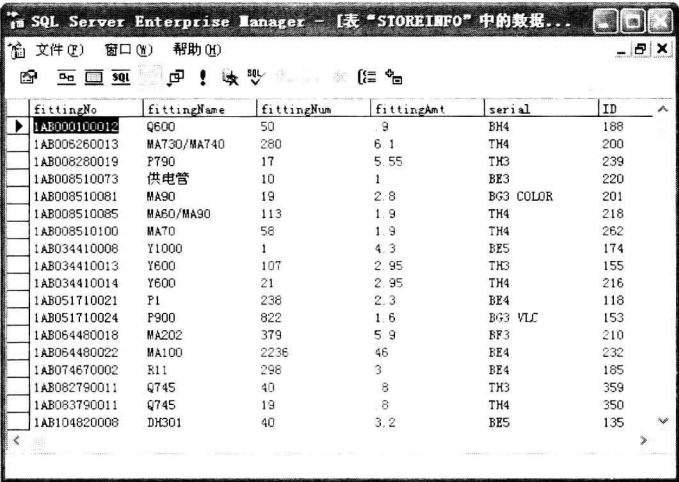


图 15-50 单击“打开表→返回所有行”



The screenshot shows the SQL Server Enterprise Manager interface with the 'STOREINFO' table selected. The table contains 20 rows of data with columns: fittingNo, fittingName, fittingNum, fittingAmt, serial, and ID.

fittingNo	fittingName	fittingNum	fittingAmt	serial	ID
1AB000100012	Q600	50	9	BH4	188
1AB006260013	MA730/MA740	280	6.1	TH4	200
1AB008280019	P790	17	5.55	TH3	239
1AB008510073	供电管	10	1	BE3	220
1AB008510081	MA90	19	2.8	BG3 COLOR	201
1AB008510085	MA60/MA90	113	1.9	TH4	218
1AB008510100	MA70	58	1.9	TH4	262
1AB034410008	Y1000	1	4.3	BE5	174
1AB034410013	Y600	107	2.95	TH3	155
1AB034410014	Y600	21	2.95	TH4	216
1AB051710021	P1	238	2.3	BE4	118
1AB051710024	P900	822	1.6	BG3 VLC	153
1AB064480018	MA202	379	5.9	BF3	210
1AB064480022	MA100	2236	46	BE4	232
1AB074670002	R11	298	3	BE4	185
1AB082790011	Q745	40	8	TH3	359
1AB083790011	Q745	19	8	TH4	350
1AB104820008	DH301	40	3.2	BE5	135

图 15-51 数据表

在该数据表中，用户可以直接修改数据表中的数据，系统会自动更新到数据库中。

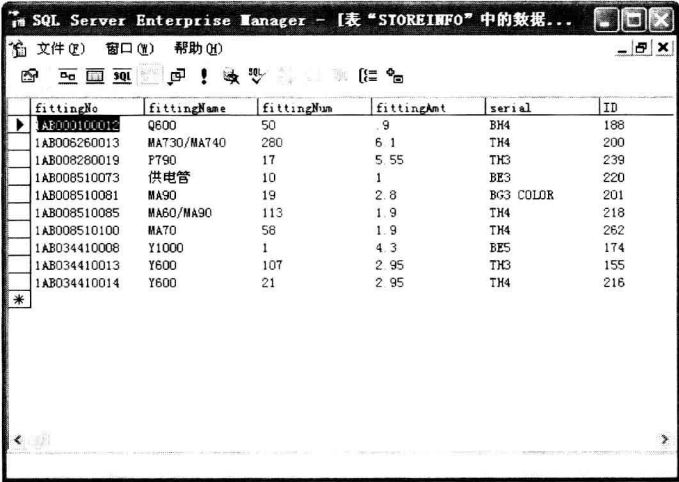
单击菜单项目“打开表→返回首行”，则程序会显示如图 15-52 所示的对话框，让用户输入要显示的最大记录行数。



The dialog box titled '行数' (Rows) contains a text field labeled '要提取的最大行数:' (Maximum number of rows to extract:) with the value '10' entered. There is a '确定' (OK) button.

图 15-52 “行数”对话框

单击“确定”按钮后，程序会显示该数据表中前若干行的记录，如图 15-53 所示。



The screenshot shows the SQL Server Enterprise Manager interface with the 'STOREINFO' table selected. The table displays the first 10 rows of data, with a scroll bar indicating more rows are available.

fittingNo	fittingName	fittingNum	fittingAmt	serial	ID
1AB000100012	Q600	50	9	BH4	188
1AB006260013	MA730/MA740	280	6.1	TH4	200
1AB008280019	P790	17	5.55	TH3	239
1AB008510073	供电管	10	1	BE3	220
1AB008510081	MA90	19	2.8	BG3 COLOR	201
1AB008510085	MA60/MA90	113	1.9	TH4	218
1AB008510100	MA70	58	1.9	TH4	262
1AB034410008	Y1000	1	4.3	BE5	174
1AB034410013	Y600	107	2.95	TH3	155
1AB034410014	Y600	21	2.95	TH4	216

图 15-53 数据表中的部分记录

单击菜单项目“打开表→查询”，则程序显示出如图 15-54 所示的 SQL 查询设计器。

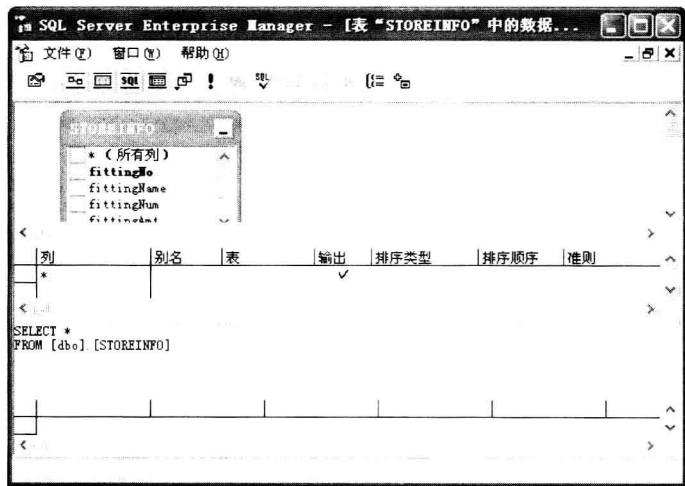


图 15-54 SQL 查询设计器

在图 15-54 中进行一些设计或直接输入 SQL 查询语句，单击工具条上的“! 运行”按钮，可执行 SQL 查询并显示出查询结果。

2. 查询分析器

在 Windows 开始菜单中单击“所有程序→Microsoft SQL Server→查询分析器”，即可运行查询分析器。

启动查询分析器，程序会显示如图 15-55 所示的“连接到 SQL Server”对话框。

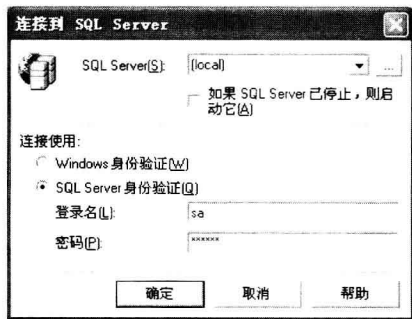


图 15-55 “连接到 SQL Server”对话框

在该对话框中输入服务器地址、用户名和密码，单击“确定”按钮即可连接服务器，然后显示如图 15-56 所示的 SQL 查询分析器主界面。

在该主界面中，左边是服务器结构树状列表，列出了服务器中所有的数据库、数据表、视图、存储过程等对象，右边是查询窗口，在上面的文本框可输入 SQL 命令语句，能以高亮度形式显示 SQL 文本，下面列出了 SQL 命令语句的执行结果。

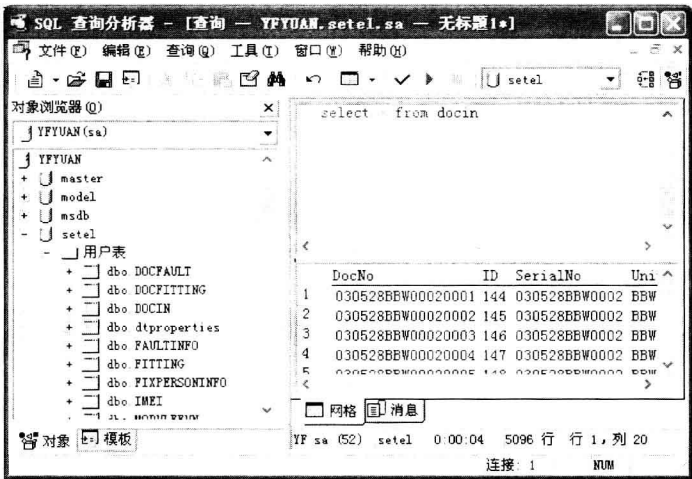


图 15-56 SQL 查询分析器主界面

(1) 执行 SQL 命令

在 SQL 文本编辑器中输入 SQL 命令文本，单击工具条上的分析查询按钮“✓”，以验证用户输入的 SQL 语句能否运行成功，但并不真正执行 SQL 命令；单击工具条上的执行查询按钮“▶”，则执行 SQL 命令，SQL 命令文本框下面会显示执行后返回的结果。

当用户选中 SQL 命令文本框中的某些文本时，分析查询或执行查询都针对用户选中的 SQL 命令文本；当用户没有选中文本时，SQL 命令文本框中所有的文本都参与操作。

在实践中，开发者经常使用 SQL 查询分析器执行 SQL 语句，比如查询数据，修改数据等。

(2) 生成 SQL 文本

在 SQL 查询分析器主界面左边的树状列表中，选中某个节点，用鼠标右击弹出快捷菜单，如图 15-57 所示。

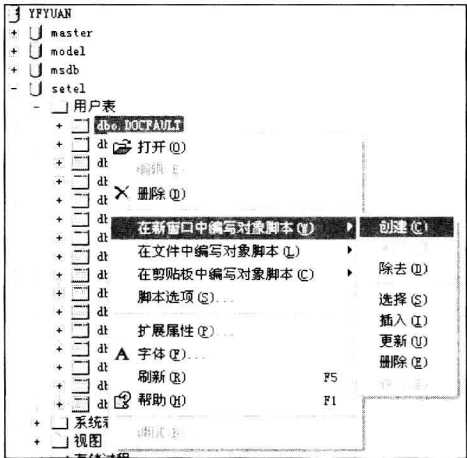


图 15-57 选中某个节点后的快捷菜单

单击菜单项目“在新窗口中编写对象脚本→创建”，程序会分析对象并自动生成创建该对象的 SQL 命令文本，如图 15-58 所示。



图 15-58 SQL 命令文本

类似地，程序还可以生成增、删、改、查数据表记录的 SQL 命令文本。

3. 事件探查器

SQL Server 事件探查器是比较高级的 SQL Server 工具，能连接服务器并实时输出服务器中执行的所有 SQL 命令文本，是一种调试应用系统的手段。

单击 Windows 开始菜单中的“所有程序→Microsoft SQL Server→事件探查器”，即可启动事件探查器程序。SQL 事件探查器的主界面如图 15-59 所示。

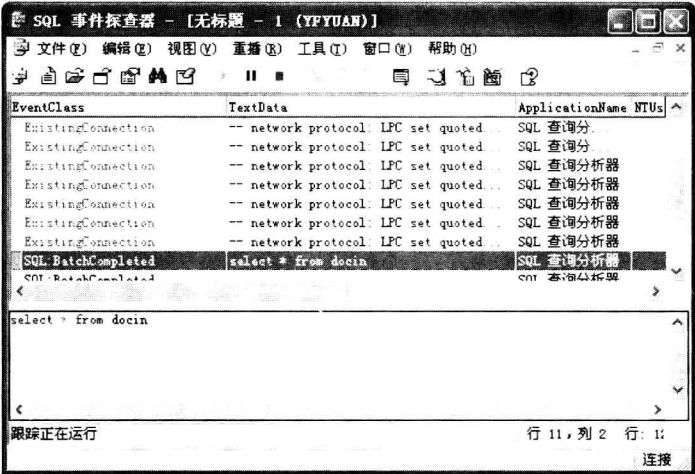


图 15-59 SQL 事件探查器主界面

下面介绍如何执行跟踪。

单击主菜单项目“文件→新建→跟踪”，程序会弹出如图 15-60 所示的“连接到 SQL Server”对话框。

在该对话框中输入服务器地址、用户名和密码，单击“确定”按钮，则程序连接服务器，然后显示如图 15-61 所示的“跟踪属性”对话框。



图 15-60 “连接到 SQL Server” 对话框

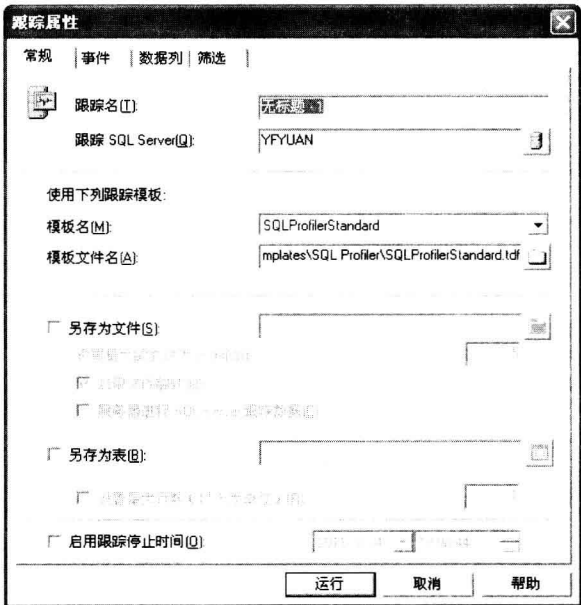


图 15-61 “跟踪属性” 对话框

在该对话框中单击“运行”按钮，关闭对话框，则事件探查器开始跟踪 SQL Server 服务器中执行的所有 SQL 命令文本，并实时地显示出来，如图 15-62 所示。

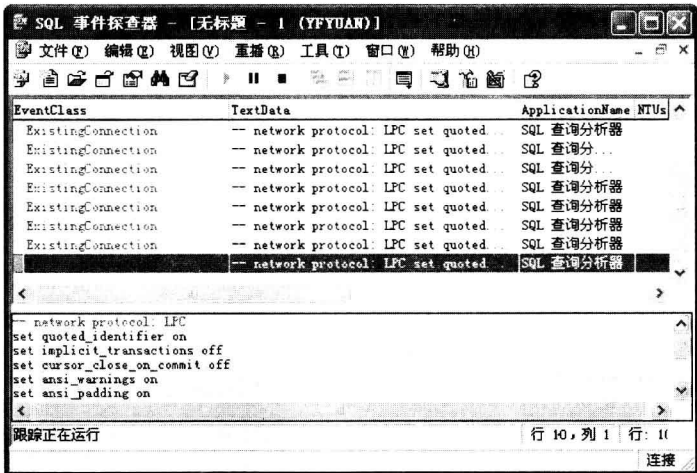


图 15-62 SQL 命令文本

比如在查询分析器或应用系统中执行 SQL 命令文本“SELECT * FROM DOCIN”，则事件探查器中会立即显示出服务器中执行的 SQL 命令文本，如图 15-63 所示。

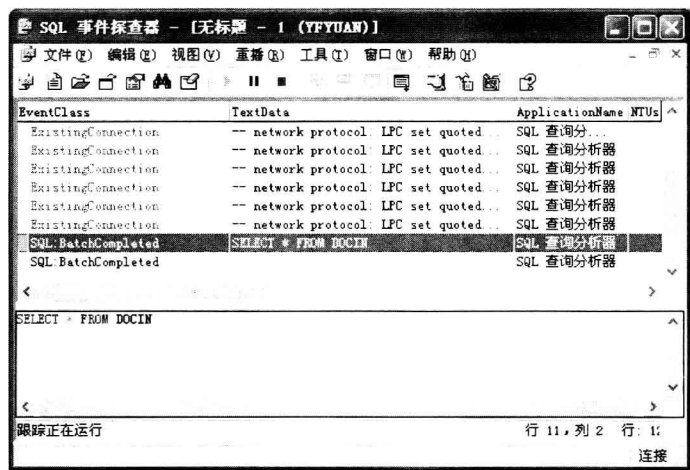


图 15-63 SQL 命令文本

单击工具条上的暂停所选择的跟踪按钮“||”，则让事件探查器暂时停止跟踪操作，此时服务器中执行的 SQL 命令文本不会显示在事件探查器中，但此前跟踪的信息还会显示在列表中。

暂停跟踪后，用户还可以单击工具条上的开始所选择的跟踪按钮“▶”，让事件探查器重新开始跟踪服务器执行的 SQL 命令。

单击工具条上的停止所选择的跟踪按钮“■”，则程序会停止跟踪操作，此时用户仍然可以单击开始所选择的跟踪按钮“▶”重新进行跟踪，但会清空跟踪列表中的内容。

4. MSDE 2000

MSDE 全称是 MS SQL Server Desktop Engine，也就是 MS SQL Server 的桌面版，是基于 MS SQL Server 核心技术构建的数据库引擎，它是免费的，与 MS SQL Server 完全兼容，可以使用企业管理器、查询分析器和事件探查器连接 MSDE 数据库。但它的功能受到限制，数据库文件最大不能超过 2GB，并发用户不超过 5 个。目前常用的是 MSDE 2000 版本。

MSDE 比 Access 强大，可以作为小型应用系统的数据库，也可用做开发过程中使用的测试数据库。

15.2 SQL 语言

SQL 语言（Structured Query Language，结构化查询语言），是一种通用的关系型数据库访问语言，能使用它来实现数据库结构的创建和维护，以及数据库记录的增、删、改、查。

SQL 语句已经成为一种国际标准，它用于描述需要执行哪些操作，而没有描述如何执行；数据库引擎会解释 SQL 语句，自行决定如何执行。这样上层的应用程序只需要通过简单的 SQL 语句告诉底层的数据库引擎想要获得什么样的数据，而数据库引擎会封装底层的复杂操作，并将

获得的数据返给上层应用程序。

通过 SQL 语言，开发者可以很方便地使用各种关系型数据库来存放大量的数据，并能降低在各个数据库系统上进行移植的工作量。

15.2.1 查询数据

SQL 语句最常用的功能就是查询数据。查询数据的 SQL 语句其常见的结构为“Select 字段列表 From 数据表名 Where 查询条件 Order By 排序方式 Group By 分组方式”，其中 Select 子语句和 From 子语句是必须的，其他都是可选的。

1. 单表查询

若数据库中有一个名为“工作单位表”的数据表，其内容如表 15-3 所示。

表 15-3 工作单位表

RecordID	单位名称	单位地址	负责人	备 注	经营范围
0	朋友	未知			
1	蓝天公司	长征路 188 号	程序猿	程序猿开发源程序	软件开发
2	绿水公司	山西路 250 号	总猿		软件设计
3	深渊技术公司	中山北路 22 号	袁总		软件实施
4	黄河大酒店	唐古拉山顶	华夏		需求调研
5	大地集团	新街口中央	周星星		房地产
6	逗你玩娱乐城	德基广场	马三爷		逗你玩
7	大富豪高端会所	殷都	纣王		各种娱乐

则可以编写只针对于这个数据表的 SQL 查询语句，获得该表格的一个子集。对于 SQL 语句“Select * From 工作单位表”，其查询结果就是这个数据表中的所有内容，这里的“*”是字段通配符，表示该数据表中所有的字段。

对于 SQL 语句“Select RecordID， 单位名称 From 工作单位表”，其查询结果如表 15-4 所示。

表 15-4 查询结果

RecordID	单 位 名 称
0	朋友
1	蓝天公司
2	绿水公司
3	深渊技术公司
4	黄河大酒店
5	大地集团
6	逗你玩娱乐城
7	大富豪高端会所

此 SQL 查询只查询“RecordID”和“单位名称”两个字段值，其他字段值不管。

对于 SQL 语句“Select * From 工作单位表 Where 单位名称 = '蓝天公司'”，其查询结果如表 15-5 所示。

表 15-5 查询结果

RecordID	单位名称	单位地址	负责人	备 注	经营范围
1	蓝天公司	长征路 188 号	程序猿	程序猿开发源程序	软件开发

数据库系统会根据其中的“Where 单位名称 = '蓝天公司'”筛选数据库的记录，只向用户提供找到的记录。

对于 SQL 语句“Select * From 工作单位表 Where RecordID < 5 Order By 单位名称”，其查询结果如表 15-6 所示。

表 15-6 查询结果

RecordID	单位名称	单位地址	负责人	备 注	经营范围
4	黄河大酒店	唐古拉山山顶	华夏		需求调研
1	蓝天公司	长征路 188 号	程序猿	程序猿开发源程序	软件开发
2	绿水公司	山西路 250 号	总猿		软件设计
0	朋友	未知			
3	深渊技术公司	中山北路 22 号	袁总		软件实施

对于这些 SQL 语句，数据库系统首先处理“Where RecordID < 5”条件，筛选数据库记录，然后再处理“Order By 单位名称”语句，对已经获得的数据按照“单位名称”字段值进行从小到大的排序，然后将排序后的结果集提供给用户。

2. 联合查询

数据库中的数据表之间存在很多逻辑关系，最常见的是主从关系，而 SQL 查询能跨越多个数据表，同时查询其中的数据。

例如数据库中有两个数据表“工作单位表”和“人员基本信息表”，这两个数据表存在主从关系，其中“人员基本信息表”的部分内容如表 15-7 所示。

表 15-7 人员基本信息表

RecordID	所属单位 ID	姓名	部 门	职 位	电 话	传 真
1	2	周伯通	市场部	销售员	(030) 355576470	0
2	2	游坦之	市场部	销售代表	(069) 202459840	(069) 202458740
3	5	黄牛	市场部	销售代表	(025) 305984100	(025) 305985110
4	0	萧何	市场部	物主	(030) 311234560	(030) 311335570
5	5	林冲	市场部	销售经理	(030) 617761100	(030) 617761110
6	6	包拯	市场部	销售员	(030) 296725420	(030) 296733330

续表

RecordID	所属单位 ID	姓名	部 门	职 位	电 话	传 真
7	5	李世民	市场部	采购员	(0241) 103912310	(0241) 105942820
8	0	慕容博	市场部	销售代理	(0217) 20124670	(0217) 20124680
9	3	无名	市场部	销售经理	(0571) 862132430	(0571) 862233440

其中“所属单位 ID”关联到数据表“工作单位表”的“RecordID”字段。

对于这种主从关系表，开发者可以使用联合查询同时查询这两张数据表中的数据，例如执行以下 SQL 语句：

```
Select
    工作单位表.单位名称,
    人员基本信息表.姓名,
    人员基本信息表.部门,
    人员基本信息表.职位
From 工作单位表,人员基本信息表
Where
    工作单位表.RecordID = 人员基本信息表.所属单位 ID
Order By 工作单位表.单位名称
```

则查询结果如表 15-8 所示。

表 15-8 查询结果

单 位 名 称	姓 名	部 门	职 位
大地集团	诸葛谨	销售部	销售代表
大地集团	留德华	市场部	销售代表
大地集团	黄牛	市场部	销售代表
大地集团	林冲	市场部	销售经理
大地集团	千里独行	市场部	市场经理
大地集团	宋江	市场部	销售代表
大地集团	李世民	市场部	采购员
大地集团	诸葛孔明	销售部	市场经理
大地集团	陈玄凤	销售部	销售经理
大地集团	黄飞虎	市场部	销售代理
大地集团	谢安	销售部	物主
大富豪高端会所	乔致庸	市场部	市场助理
大富豪高端会所	云中鹤	市场部	物主
大富豪高端会所	令狐冲	市场部	市场经理

在这个 SQL 语句中，Select 子语句列出了不同表的字段，From 子语句列出了参与查询的数据表名，Where 子语句定义了数据表之间的字段值关系。

3. 函数

开发者可以在 SQL 语句中调用一些函数，这些函数从来源上可以分为三类：

- (1) 标准函数。这是在 ANSI SQL 标准中定义的函数，是所有的数据库系统都支持的，因此可以放心大胆地使用。
- (2) 系统特定函数。这是由特定的数据库系统支持的函数，不同的数据库系统会支持不同的函数集，因此使用这些数据库系统特定的函数会影响应用系统的可移植性。
- (3) 用户自定义函数。一些数据库系统支持用户自己定义函数，但很多小型数据库系统不支持。

函数从功能上可以分为合计函数和标量函数两种。

(1) 合计函数

合计函数处理多个数据并返回单一的处理结果值，函数参数为多个字段值。在开发中常用的合计函数如表 15-9 所示。

表 15-9 合计函数

合计函数	功 能
Count	这个函数用于累计数据库记录的个数，查询结果是一个整数。当用做“Count(*)”时返回所有记录的个数，当用做“Count(字段值)”时返回记录的个数，但当字段值为 NULL 时不参与累计 例如“Select Count(*) From 工作单位表”，它用于查询数据表“工作单位表”中所有数据库记录的个数
Max	用于获得字段值的最大值 例如“Select Max(RecordID) From 工作单位表”，它用于查询数据表“工作单位表”的字段“RecordID”的最大值，查询结果的数据类型和 RecordID 字段的数据类型一样
Min	获得字段值的最小值 例如“Select Min(RecordID) From 工作单位表”，它用于查询字段“工作单位表.RecordID”的最小值
Avg	获得字段值的算术平均值
Sum	获得字段值的总和

例如，有一个名为 Orders 的数据表，其内容如表 15-10 所示。

表 15-10 Orders 数据表

OrderID	OrderDate	OrderPrice	Customer
1	2008/12/29	1000	[NULL]
2	2008/11/23	1600	Carter
3	2008/10/05	700	Bush
4	2008/09/28	300	Bush
5	2008/08/06	2000	Adams
6	2008/07/21	100	Carter

对该数据表执行 SQL 语句，其查询结果如表 15-11 所示。

表 15-11 查询结果

SQL 语句	查 询 结 果
Select Max(OrderPrice) From Orders	2000
Select Min(OrderPrice) From Orders	100
Select Avg(OrderPrice) From Orders	950
Select Sum(OrderPrice) From Orders	5700
Select Count(*) From Orders	6
Select Count(Customer) From Orders	5 （由于第一条记录的 Customer 字段值为 NULL，因此不参与累计）

（2）标量函数

标量函数处理一个数据并返回一个结果值，函数参数为单个字段值。MS SQL Server 支持 Len 函数，能返回字符串的长度。例如上面的 Orders 数据表，执行 SQL 语句“Select CustoeMr , Len(Customer) As Length From Orders”，其查询结果如表 15-12 所示。

表 15-12 查询结果

Customer	Length
[NULL]	0
Carter	6
Bush	4
Bush	4
Adams	5
Carter	6

常用的标量函数如表 15-13 所示。

表 15-13 常用的标量函数

标 量 函 数	功 能
Abs	获得数值的绝对值
Sin	计算数值的 SIN 值
Cos	计算数值的 COS 值

每个数据库管理系统支持的函数集有着比较大的不同，因此在具体应用时需要参考当前数据库管理系统的用户手册，查找所需的函数及其用法。

15.2.2 新增数据

在 SQL 语言中 Insert 语句用于新增数据库记录，其用法为“Insert Into 表名(字段列表) Values(数值列表)”。

例如对于前面示例的“工作单位表”，可以编写以下 SQL 语句来向数据库新增一条记录。

```
Insert Into 工作单位表
( RecordID , 单位名称 ,单位地址,负责人 )
```

```
Values  
( 20 , '天海公司', '珠江路 100 号', '张三' )
```

在这个 SQL 语句中，并没有列出数据表中所有的字段，只设置新增记录的部分字段的值，其他字段使用默认值。

这里 SQL 语句中的字符串值是用单引号括起来的，若字符串值有单引号字符，则使用连续的两个单引号字符进行转义。

注意，在某些数据库系统中是使用双引号定义字符串值的。

对于 MS SQL Server 或 Access 数据库的 SQL 语句，可以使用“0x 十六进制字符串”来表示二进制数据。例如使用“Insert PictureTable (Name , Picture) Values ('file.bmp', 0x5A9B8FC83E……)”语句来向数据库插入该二进制字段值，这里的二进制字段值没有使用单引号括起来。

15.2.3 修改数据

在 SQL 语言中使用 Update 语句来修改数据库中已有的记录，其用法为“Update 表名 Set 字段名 1=数值 1，字段名 2=数值 2 Where 查询条件”。

例如对于前面示例的“工作单位表”，可以使用以下 SQL 语句来修改已有的记录。

```
Update 工作单位表 Set  
    单位名称 = '地龙公司',  
    单位地址 = '珠江路 100 号',  
    , 负责人 = '李四',  
Where RecordID = 20
```

执行这个 SQL 语句能修改数据表中字段 RecordID 的值为 20 的所有记录的字段值。

在这个 SQL 语句中，并没有列出所有的字段，只修改了部分字段的值，其他字段值保持不变。

特别要注意，编写 SQL 语句时要仔细设置查询条件，在这里如果没有“Where RecordID = 20”，则系统会修改数据表“工作单位表”中所有记录的字段值，很容易造成灾难性的后果。

15.2.4 删除数据

在 SQL 语言中使用 Delete 语句删除数据库记录，其用法为“Delete From 表名 Where 查询条件”。

例如对于“工作单位表”，可以使用以下 SQL 语句来删除已有的记录。

```
Delete From 工作单位表  
Where RecordID = 20
```

执行这个 SQL 语句能删除数据表中字段 RecordID 的值为 20 的所有记录。

特别要注意，编写 SQL 语句时要仔细设置查询条件，在这里如果没有“Where RecordID = 20”，则系统会删除数据表“工作单位表”中所有的记录，很容易造成灾难性的后果。

15.2.5 视图

视图（View）就是数据库中预定义的由 SQL 查询结果组成的虚拟数据表，开发者可以使用 SQL 语句在这个虚拟表上进行二次查询。在编写 SQL 语句时，视图可以当做数据表来使用。

很多数据库系统支持视图，MS Access 也支持视图。

例如，MS SQL Server 中预定义了一个名为“INFORMATION_SCHEMA.TABLES”的视图，若能返回数据库中所有数据表的定义信息，则执行 SQL 语句“Select * From INFORMATION_SCHEMA.TABLES”的查询结果如表 15-14 所示。

表 15-14 查询结果

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
销售管理	dbo	dtproperties	BASE TABLE
销售管理	dbo	sysconstraints	VIEW
销售管理	dbo	syssegments	VIEW
销售管理	dbo	Table_AppCategory	BASE TABLE
销售管理	dbo	Table_AppCategoryValue	BASE TABLE
销售管理	dbo	Table_EntryRelation	BASE TABLE
销售管理	dbo	产品型号表	BASE TABLE
销售管理	dbo	订购机型表	BASE TABLE
销售管理	dbo	费用信息表	BASE TABLE
销售管理	dbo	附属文件表	BASE TABLE
销售管理	dbo	工作单位表	BASE TABLE
销售管理	dbo	工作日志表	BASE TABLE
销售管理	dbo	客户访问记录表	BASE TABLE
销售管理	dbo	人员基本信息表	BASE TABLE
销售管理	dbo	特殊日期表	BASE TABLE
销售管理	dbo	项目基本信息表	BASE TABLE

程序可以通过视图来更新和删除数据库记录，但受到一些限制。

使用视图具有以下优点：

- （1）视图能将系统中某些分散的数据集中起来，只列出用户关心的数据，其他无用的数据不显示。
- （2）简化操作。视图是预定义的 SQL 查询，因此用户只需调用视图而无须编写 SQL 语句，可以简化操作。
- （3）定制数据。可以根据需要为不同的用户定制视图，使数据库中相同的数据结构对不同的用户显示出不同的结构。
- （4）安全性。视图可以作为一种安全机制，系统管理员可以给用户分配访问视图的权限。用

户只能通过视图来访问特定的数据，而没有能力访问其他数据。

15.2.6 存储过程

存储过程（Stored Procedure）是一组实现了特定功能的 SQL 语句集合，它可以看做一个使用 SQL 语句编写的函数，有参数，有返回值。这个函数已经保存在数据库系统中，应用程序可以通过 SQL 语句来调用它。

大型数据库系统支持存储过程，但小型数据库系统可能不支持，比如 MS Access 就不支持存储过程。

应用程序可以通过 SQL 语句来调用存储过程，其语法为“存储过程名 参数 1,参数 2,参数 3”。在存储过程内部可以进行数据库记录的增、删、改、查操作，然后返回操作结果。返回的可以是一个值或二维表格。

例如对于 MS SQL Server 有一个名为“sp_columns”的存储过程，若执行 SQL 语句“sp_columns '人员基本信息表'”，则查询结果如表 15-15 所示，此处没有列出所有的查询字段。

表 15-15 查询结果

TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	TYPE_NAME
销售管理	dbo	人员基本信息表	RecordID	int
销售管理	dbo	人员基本信息表	所属单位 ID	int
销售管理	dbo	人员基本信息表	姓名	nvarchar
销售管理	dbo	人员基本信息表	部门	nvarchar
销售管理	dbo	人员基本信息表	职位	nvarchar
销售管理	dbo	人员基本信息表	电话	nvarchar
销售管理	dbo	人员基本信息表	传真	nvarchar
销售管理	dbo	人员基本信息表	手机号	nvarchar
销售管理	dbo	人员基本信息表	备注	ntext
销售管理	dbo	人员基本信息表	电子信箱	nvarchar
销售管理	dbo	人员基本信息表	QQ 号	nvarchar
销售管理	dbo	人员基本信息表	特殊纪念日	nvarchar

通过调用存储过程，程序就能获得数据表中所有字段的详细信息而无须查询系统表。

在业界关于存储过程的应用有两种思想，第一种思想是应用程序应该尽可能广泛地使用存储过程，甚至将一些业务逻辑的功能也做到存储过程中。其好处是能提高程序性能，而且能通过调整存储过程来修改应用系统的行为，而无须修改应用系统的程序。

第二种思想是应用程序尽可能不使用存储过程，尽量在程序中使用标准的 SQL 语句来操作数据库。其好处是程序的可移植性比较好，能方便地切换不同的数据库而运行，但使用 SQL 语句的性能有时确实赶不上存储过程。

做产品类软件研发工作的，建议用第二种思想，因为产品类软件要求可移植性比较好，要求

能在各种数据库上运行。对于从事特定的项目类软件开发工作的，因为没有数据库系统移植的需求，因此可以考虑用第一种思想。

15.2.7 触发器

触发器（Tigger）是一种特殊的存储过程，它的执行不是应用程序或手动启动执行的，而是由特定的事件触发的。这相当于使用 SQL 语句编写数据库的事件处理。

例如可以对某个数据表添加删除记录的触发器，当以任何方式，比如应用程序、存储过程、管理员手工操作等，删除了该数据表的记录时，系统会自动调用删除记录触发器，执行其中的 SQL 代码，完成特定的功能。

触发器可能会被频繁地调用，因此会影响数据库的性能，要慎重使用触发器。

一些小型数据库系统不支持触发器，例如 MS Access。

第 16 章

商业软件开发规范

在商业软件开发中，编写代码是其中的一个比较重要的部分。为了软件代码的可维护性，开发人员在编写代码时需要遵守一些代码书写原则，潦草的代码是难以阅读、维护和交流的。代码书写原则和代码的语法是相关的，下面针对一些语法说明其代码书写原则。

16.1 C#代码书写规范

统一的代码书写规范是一个组织进行开发团队管理的基础手段，不管是高级开发人员还是普通程序员都应当遵守相同的 C#代码书写规范。不过很多组织和个人在长期的软件开发实践中形成了不尽相同的代码书写规范，而且个人可能在多次更换工作单位的时候受到不同风格的代码书写规范的影响，难于统一。但是这项工作还得进行，而且个人的规范不能与组织的规范冲突。

在实践中不断进化出来的比较好的代码书写规范之间是大致相同的，在此推荐一种 C#代码书写规范。

16.1.1 代码缩进

程序块需要采用代码缩进风格编写，缩进的长度为 4 个空格，方式为 IDE 自动缩进或使用 TAB 键缩进。一行代码字符长度（含空格，汉字算两个字符）不能超过 110 个，最好不超过 80 个，尽量避免一行代码写得过长的情况。开发工具自动生成的代码除外。

函数或过程的开始、结构的定义、循环、判断语句等代码都需要采用缩进风格，case 语句下的功能性代码也需要采用缩进风格，此外 try-catch-finally、using() 语法结构都需要采用代码缩进。

例如以下代码就是符合代码缩进规范的。

```
string strNumber1 = "123";
string strNumber2 = "456";
string strOperator = "+";
try
{
    double number1 = Convert.ToDouble( strNumber1);
    double number2 = Convert.ToDouble( strNumber2 );
    double result = 0;
    switch (strOperator )
```

```

    {
        case "+":
            result = number1 + number2;
            break;
        case "-":
            result = number1 - number2;
            break;
        case "*":
            result = number1 * number2;
            break;
        case "/":
            result = number1 / number2;
            break;
    }
    MessageBox.Show( result.ToString());
}
catch (Exception ext)
{
    MessageBox.Show(ext.Message);
}

```

16.1.2 空行

空行用于将与逻辑无关的代码段分隔开，以便提高可读性。

以下情况推荐使用两个空行：

- (1) 一个源文件中的两个片段（session）之间。
- (2) 类声明和接口声明之间。

以下情况推荐使用一个空行：

- (1) 两个方法之间。
- (2) 方法内局部变量和方法的第一条功能性语句之间。
- (3) 块注释（使用/* */）或单行注释之间。
- (4) 一个方法内的两个逻辑段之间。

16.1.3 换行

当一条语句超出或即将超出建议的列宽时，应当进行换行以维持列宽限制。换行规则如下：

- (1) 在逗号后面分行。
- (2) 在运算符、关键字前换行。
- (3) 在较高基本运算符处换行。
- (4) 新的一行应该与上一行同级别表达式的开头处对齐。

(5) 如果以上规则导致代码混乱，或者代码都堆积到右边，则可以使用缩进 8 个字符代替，若这样还不行则自行处理。

- (6) 规则 1 优先于规则 2。

【范例 1】当调用方法时的代码行太长时，可以进行以下换行。对于方法参数若进行换行，则每个参数代码占据一行，即使参数代码比较短也要占据一行。

```
DialogResult result2 = MessageBox.Show(  
    null,  
    "你输入的年龄不正确，应该是大于 0 小于 150，请重新输入!",  
    "系统提示",  
    MessageBoxButtons.OK,  
    MessageBoxIcon.Information);
```

【范例 2】下面的 Drawing 函数参数太多，应当换行并缩进，其中的 RectangleF 类型的构造函数参数也比较多，因此进行二次换行缩进。

```
g.DrawString(  
    "标题",  
    new System.Drawing.Font("宋体", 9),  
    System.Drawing.Brushes.Red,  
    new System.Drawing.RectangleF(  
        left,  
        top,  
        right - left,  
        bottom - top));
```

【范例 3】下面定义变量时，变量类型名称太长，需要在“=”后面换行缩进；if 语句的判断表达式太长，需要在运算符前进行换行缩进。

```
System.Drawing.ContentAlignment align  
    = System.Drawing.ContentAlignment.BottomLeft;  
System.Drawing.StringAlignment align2  
    = System.Drawing.StringAlignment.Near;  
if (align == System.Drawing.ContentAlignment.TopCenter  
    || align == System.Drawing.ContentAlignment.MiddleCenter  
    || align == System.Drawing.ContentAlignment.BottomCenter)  
{  
    align2 = System.Drawing.StringAlignment.Center ;  
}
```

对于上面的判断表达式也可以写成如下的换行方式。

```
if (align == System.Drawing.ContentAlignment.TopCenter ||  
    align == System.Drawing.ContentAlignment.MiddleCenter ||  
    align == System.Drawing.ContentAlignment.BottomCenter)  
{  
    align2 = System.Drawing.StringAlignment.Center ;  
}
```

不过还是推荐使用运算符在行首的换行方式。

【范例 4】在下面的 for 语句中，由于太长，进行换行缩进，for 语句中的分号放在行尾。

```
byte[ ] sourceArray = new byte[100] ;  
byte[ ] destArray = new byte[300];  
for (int sourceIndex = 0, destIndex = 100;  
    sourceIndex < sourceArray.Length && destIndex < destArray.Length;  
    sourceIndex++, destIndex++)  
{  
    destArray[destIndex] = sourceArray[sourceIndex];  
}
```

16.1.4 空格

要遵循手写英文中标点符号的规范，字符逗号“,”和分号“;”后要加空格，圆括号“(”除外，例如：

```
Background = new Color( red , green , blue ) ;
```

此外在操作符前后加上空格可增强代码的可读性，避免代码纠缠在一起。

对于与英文意义不同的操作符不遵循这个规则，例如“.”符号前后不用空格。

16.1.5 定义类型

在定义每个类型或成员时都必须明显地指定可访问级别，必须使用“public”、“protected”、“private”或“internal”关键字进行修饰。定义类型的基础类型时，首先写出它的基础类型，然后再写出它所实现的接口。

定义类型成员时，各个成员尽量按如下顺序进行排序：

- (1) 静态构造函数。
- (2) 静态字段、方法。
- (3) 类型的构造函数和析构函数。
- (4) 字段。
- (5) 属性。
- (6) 公开的方法。
- (7) 私有的方法。

功能相近或相关的成员尽量放在一起，尽量按照成员的名称或功能排序。

例如，以下代码定义了一个类型，类型成员符合上述规范。

```
public class PeopleClass :  
    System.ComponentModel.Component ,  
    System.Collections.IComparer  
{  
    static PeopleClass()  
    {  
        Console.WriteLine("start peopleclass");  
    }  
  
    public static int InstanceCount = 0;  
  
    public PeopleClass()  
    {  
        InstanceCount++;  
    }  
  
    private string _Name = null;
```

```
public string Name
{
    get { return _Name; }
    set { _Name = value; }
}

private string _Address = null;

public string Address
{
    get { return _Address; }
    set { _Address = value; }
}

public int Compare(object x, object y)
{
    throw new NotImplementedException();
}
```

在这段代码中属性“Name”表示人员姓名，而“Address”表示地址的意思。一般来说，提到一个人首先想到的是姓名，然后才是地址，因此虽然按照英文排序 Address 应该排在 Name 之前，但此处按照功能将 Name 排在 Address 前面。

需要注意的是，对于“私有字段-公开属性”成员对的安排有两种方式，一种是上面的字段和对应的属性放在一起，如以下代码：

```
private string _Name = null;

public string Name
{
    get { return _Name; }
    set { _Name = value; }
}

private string _Address = null;

public string Address
{
    get { return _Address; }
    set { _Address = value; }
}
```

另一种是字段值和属性集中在一起，如以下代码：

```
private string _Name = null;
private string _Address = null;

public string Name
{
    get { return _Name; }
    set { _Name = value; }
}

public string Address
{
    get { return _Address; }
    set { _Address = value; }
}
```


这两种方式都可行，建议将私有字段和对应属性放在一起，因为这两个成员功能上是紧密关联的，这样便于完整地修改和删除代码块。

此外业界还流行以下写法：

```
private string address = null;

public string Address
{
    get { return address; }
    set { address = value; }
}
```

也就是字段名和属性名一样，只是字段名开头小写，而属性名开头大写。不建议用这种写法，因为当在类型内部引用属性时，很容易出现难以察觉的程序错误，有可能导致想要引用字段时结果误用了属性，而引用属性时结果误用了字段。代码书写规范是帮助写出没有错误的代码的，而不应该留下种种隐患。

16.1.6 大小写

标识符中所有的字母都应大写，仅对于由两个或更少字母组成的标识符不使用该约定。例如，System.IO，System.Web.UI。

C#中采用两种大小写规则：

Pascal 规则：除了保留字和指示符是小写的外，其他所有的标识符采用驼峰格式，即每个标识符的开头字母大写，内嵌单词的首字母也要大写，只取首字母的缩写词也一样。

Camel 规则：首个单词的首字母小写，内嵌单词的首字母大写。

此外在开发中还有可能遇到匈牙利规则，这个规则的基本原则是：变量名=属性+类型+对象描述，其中每一个对象的名称都要求有明确含义，可以取对象名全称或名字的一部分，命名要易于容易记忆和理解的原则。

表 16-1 汇总了大写规则，并提供了不同类型的标识符示例。

表 16-1 大写规则

标 识 符	大 小 写	示 例
类	Pascal	AppDomain
枚举类型	Pascal	ErrorLevel
枚举值	Pascal	FatalError
事件	Pascal	TextChanged
异常类	Pascal	WebException
只读静态字段	Pascal	MyClass.Instance
接口	Pascal	IDisposable
方法	Pascal	ToString
属性	Pascal	BackColor

续表

标 识 符	大 小 写	示 例
公共实例字段	Pascal	MyClass.RedValue
受保护的实例字段	Camel	MyClass.redValue
私有实例字段	Camel	MyClass.redValue
参数	Camel	typeName
方法内的变量	Camel	tempValue

16.1.7 名称

对于个人或组织，对已有的命名风格应该保持一致，而且个人风格不能与组织风格冲突。用正确的反义词对来命名具有互斥意义的类型、变量或方法。

下面就是一些常用的反义词对：

add / remove	begin / end	create / destroy	insert / delete
first / last	get / release	get / set	increment / decrement
put / get	add / delete	lock / unlock	open / close
max / min	old / new	start /stop	next / previous
source / target	show / hide	send / receive	source / destination
cut / paste	up / down		

例如：

```
int min_sum ;
int max_sum ;
int AddUser ( string name ) ;
int DeleteUser ( string name ) ;
```

对于特定的类型还有一些额外的命名规则。

（1）对于从 `System.Attribute` 类型派生的类型，其名称必须以 “Attribute” 结尾，例如 `BrowsableAttribute`、`STAThreadAttribute` 类型。

（2）所有的接口类型以大写的字母 I 开头，字母 I 后面的部分按照 `Pascal` 规则进行命名，例如 `IFormatProvider` 类型。

（3）对于实现了接口的类型，其类型名称建议以接口类型的名称去掉前面大写字母 I 的部分来结尾。例如 `DbDataReader` 类型实现了接口 `IDataReader`。

（4）对于委托类型，其名称建议以 “Handler” 结尾，例如 `EventHandler`，`CancelEventHandler`。

（5）对于一些基础类型，其派生类型的名称建议以基础类型的名称或部分名称结尾。例如 `Exception` 类型派生了 `InvalidOperationException` 类型，`EventArgs` 类型派生了 `CancelEventArgs` 类型，`TypeConverter` 类型派生了 `ColorConverter` 类型。

(6) 不要使用特殊的标识符, 比如 C# 的关键字的某种大小写形式, 不要使用 System、Collections、Forms、UI 等系统命名空间名称。建议不应使用 VB.NET、Delphi.NET 等其他语言的关键字, 因为使用 C# 开发的软件组件未来可能会被其他语言调用, 若使用其他语言的关键字作为类型名称, 则会导致在其他编程语言中用不了。

例如, 在 C# 中定义了一个名为 “AddHandler” 的类型, 其他 C# 程序能正常调用, 但由于 AddHandler 是关键字, 因此在 VB.NET 代码中无论如何也不能调用这个 AddHandler 类型。

以下列出了一些不应使用的标识符。

AddHandler	AddressOf	Alias	And	Ansi
As	Assembly	Auto	Base	Boolean
ByRef	Byte	ByVal	Call	Case
Catch	CBool	CByte	Cchar	CDate
CDec	CDBl	Char	Cint	Class
CLng	CObj	Const	Cshort	CSng
CStr	CType	Date	Decimal	Declare
Default	Delegate	Dim	Do	Double
Each	Else	Elseif	End	Enum
Erase	Error	Event	Exit	ExternalSource
False	Finalize	Finally	Float	For
Friend	Function	Get	GetType	Goto
Handles	If	Implements	Imports	In
Inherits	Integer	Interface	Is	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next	Not
Nothing	NotInheritable	NotOverridable	Object	On
Option	Optional	Or	Overloads	Overridable
Overrides	ParamArray	Preserve	Private	Property
Protected	Public	RaiseEvent	ReadOnly	ReDim
Region	REM	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared	Short
Single	Static	Step	Stop	String

Structure	Sub	SyncLock	Then	Throw
To	True	Try	TypeOf	Unicode
Until	volatile	When	While	With
WithEvents	WriteOnly	Xor	Eval	extends
instanceof	package	var		

在开发 WinForm 或 ASP.NET 程序时，需要给用户界面上的控件设个名称，控件名称一般采用“前缀+表示功能的标识符”的格式。表 16-2 列出了控件名称前缀。

表 16-2 控件名称前缀

控 件 类·型	控件名称前缀
Label	lbl
Button	btn
ImageButton	imgbtn
ListBox	lst
DataList	dl
CheckBoxList	chkls
RadioButtonList	rdolt
Panel	pnl
AdRotator	ar
RequiredFieldValidator	rfv
RangeValidator	rv
ValidatorSummary	vs
TextBox	txt
LinkButton	lnkbtn
DropDownList	ddl
DataGrid	dg
CheckBox	chk
RadioButton	rdo
Image	img
Calender	cld
Table	tbl
CompareValidator	cv
RegularExpressionValidator	rev

16.1.8 名称空间

名称空间的名称一般采用“公司名称.技术名称.[功能].[设计].[软件模块名称]”的格式，例如 Sky.Business。

此外，命名空间和类型不能使用相同的名称。例如某类型名为 `Debug`，则不能再使用 `Debug` 作为命名空间的名称。

16.1.9 语句

C#程序是由一条条语句组成的，对于语句推荐的书写规范如下。

1. 简单语句

对于简单语句，每行至多包含一条语句，例如：

```
number ++ ;
number -- ;
```

应当尽量避免一行包含多条语句，例如：

```
number ++ , number2 -- ;
```

2. 复合语句

复合语句是包含在大括号中的语句序列，例如“{一些语句}”。被包括在其中的语句应该较复合语句缩进一个层次。其规则有：

(1) 左大括号“{”应由复合语句首行另起一行并与其对齐；右大括号“}”应另起一行并与复合语句首行对齐。

(2) 大括号可以被用于所有的语句，包括单个语句，只要这些语句是诸如 `if-else` 或 `for` 循环体。这样便于添加语句而无须担心由于忘了加大括号而引起难于察觉的程序错误。

例如应该采用下面的写法，即使循环体中只有一条语句。

```
for (int iCount = 0; iCount < nums.Length; iCount++)
{
    nums[iCount] = iCount;
}
```

而不推荐采用下面的写法，虽然这种写法能完成功能，但却容易成为一个陷阱。

```
for (int iCount = 0; iCount < nums.Length; iCount++)
    nums[iCount] = iCount;
```

16.1.10 注释

一个便于阅读的源代码应该添加适当的注释，在 C#中添加注释推荐的规范如下。

1. 文件头注释

源代码文件头部应该添加注释，注释需要列出版本说明、创建时间、作者、内容、功能、与其他文件的关系及修改日志等。

2. 文档注释

C# 提供文档注释，在声明类型及其成员时都应尽量加上文档注释，以方便他人阅读和调用代码。

16.1.11 代码文件目录结构

一个 C# 工程中包含若干个文件或目录，典型的文件和目录如下：

(1) `image` 子目录。用于放置所有的图片文件，比如应用程序开发过程中的 `bmp`、`png`、`gif`、`ico` 文件等。

(2) `Lib` 子目录。用于放置程序开发中所有的第三程序集，比如程序引用的 `dll`、`exe` 文件。微软 .NET 框架自带的标准程序集不需要加进去。

16.2 C# 软件开发原则

下面介绍一些 C# 软件开发原则，这些原则是开发稳定、强壮的软件时所需要遵守的基本原则，是开发人员的基本功。

16.2.1 尽晚创建，尽早释放

根据实践，在软件开发中对于非托管资源应当采用“尽晚创建，尽早释放”的原则，而对于托管资源在不影响程序性能的前提下也尽量采用这种方式。

所谓“尽晚创建，尽早释放”就是指应用程序在不影响完成功能的前提下尽可能晚地创建一些占据重要系统资源的对象，尽可能早地销毁这些对象，从而释放资源。这样能尽量缩短应用程序占用重要系统资源的时间。

对于商业软件，稳定压倒一切。根据实践，软件不稳定的最大原因就是资源泄露。比如内存占有了而未释放，文件打开了而没有关闭，虽然 .NET 框架提供了很好的垃圾回收机制，但这种回收机制仅限于处理托管资源。而开发程序时会使用大量的非托管资源，比如文件、数据库连接、网络连接等。对于非托管资源需要采用“尽晚创建，尽早释放”的原则。

在 .NET 开发中，使用了非托管资源的常见类型主要有：

(1) 流对象。流对象类型是从 `System.IO.Stream` 类型派生的，大多数都在 `System.IO` 名称空间下，最常见的是 `System.IO.FileStream` 类型。

(2) 数据库连接。数据库连接对象都实现了 `System.Data.IDbConnection` 接口，大多数都在 `System.Data` 名称空间下，比如 `System.Data.Odbc.OdbcConnection`、`System.Data.SqlClient.SqlConnection`、`System.Data.OleDb.OleDbConnection` 等类型。

(3) 图形图像对象。名称空间 `System.Drawing` 下面有很多类型都是与图形相关的，它们都用了非托管对象，其内部使用了各种与图形相关的 Win32 句柄，比如 `Graphics`、`Pen`、`Brush` 类型等。

(4) WinForm 控件类型。WinForm 控件类型都是 `System.Windows.Forms.Control` 的派生类型，大多数安排在 `System.Windows.Forms` 名称空间下，比如 `System.Windows.Forms.TextBox` 类型。由于大多数程序的控件都是使用 WinForm 窗体设计器设计出来的，设计器已经自动生成了加载和销毁控件的代码，因此一般的开发人员无须考虑它们的资源泄露问题。

在软件代码编写中，“尽晚创建、尽早释放”是随时都要记住和遵守的规则。当软件需要使用非托管资源时，一般要在确定使用非托管资源前一步创建它，当确定不再使用该托管资源，或者短期内不再使用该托管资源时，需要立即释放这些资源。

例如，以下代码遵循了“尽晚创建，尽早释放”的原则。

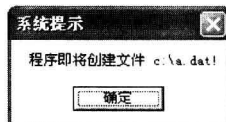
```
//确定文件名
string fileName = "c:\\a.dat";
//显示一个消息框提醒用户
MessageBox.Show("程序即将创建文件 " + fileName + "!", "系统提示");
//打开文件
System.IO.StreamWriter stream = new System.IO.StreamWriter(fileName, false);
//写入文件内容
stream.Write("文件创建时间:" + DateTime.Now.ToString());
//关闭文件
stream.Close();
//显示一个消息框提醒用户
MessageBox.Show("程序成功创建文件 " + fileName + "!", "系统提示");
```

而以下的代码没有遵循该原则：

```
//确定文件名
string fileName = "c:\\a.dat";
//打开文件
System.IO.StreamWriter stream = new System.IO.StreamWriter(fileName, false);
//显示一个消息框提醒用户
MessageBox.Show("程序即将创建文件 " + fileName + "!", "系统提示");
//写入文件内容
stream.Write("文件创建时间:" + DateTime.Now.ToString());
//显示一个消息框提醒用户
MessageBox.Show("程序成功创建文件 " + fileName + "!", "系统提示");
//关闭文件
stream.Close();
```

在上面的代码中，代码块 `MessageBox.Show("程序即将创建文件 " + fileName + "!", "系统提示");` 用于显示如图 16-1 所示的对话框。

只有用户单击“确定”按钮关闭对话框后，系统才会执行后续的程序代码，而等待用户单击“确定”按钮的时间是不一定的，可能是几秒，也可能是几分钟甚至几个小时。



在第一段代码中，程序在等待用户关闭消息对话框后才打开

图 16-1 “系统提示”对话框

文件，写入数据，关闭文件。整个过程速度很快，不会超过 1 秒钟。

而在第二段代码中，程序早早地打开文件，然后显示消息对话框，在等待用户关闭消息对话框的漫长的时间中，程序始终维持着打开文件的状态，只有等到用户关闭消息对话框后才往文件中写数据，然后再显示出一个消息框等待用户确定，最后才关闭文件。由于第二段代码长期占据重要的文件句柄资源而不使用，从其他程序代码的角度看这是很自私的行为，完全不考虑其他程序代码的工作。

有人或许要问，现在计算机硬件很廉价，为了开发速度，程序浪费些资源不会有多大问题。这种想法是不对的，主要有以下两个原因。

(1) 古训：千里长堤毁于蚁穴。现代软件的很多缺陷和错误都是一个个很微小的缺陷和错误堆积出来的。一些软件会使用循环处理大量的数据，可能会循环上百万次，此时循环中的一个很微小的缺陷很可能导致软件的严重缺陷。软件开发人员在平时编写代码时遵循一些原则就可以避免一些很微小的缺陷和错误，降低软件的缺陷，提高软件的质量。

(2) 现在的操作系统都是多进程多线程的。因此计算机不是属于某个软件私有的，是多个软件共享的，很多系统资源，比如文件、数据库连接等都可能是多个软件共享的。当软件不遵守“尽早创建，尽早释放”的原则，应用程序长期地没有理由地蛮横占据着一些重要的系统资源而不使用时，这是一种“流氓”行为，可能会影响其他软件的正常运行。计算机软件没有高低贵贱之分，都根据功能需要而占用合适的资源。若一个软件流氓行迹比较重，严重影响其他软件的运行，造成整个系统不稳定，则很容易让用户认为该软件不可靠，会带来麻烦，这种软件是用户所讨厌的。

16.2.2 单入口，单出口

在编写一个类型成员方法时，应当尽量遵守单入口单出口的原则。所谓单入口单出口的原则就是指从唯一的地方进入方法体，从唯一的地方退出方法。在实现上就是方法体最后一行出现 `return` 语句，其他地方没有 `return` 语句。而与之对应的是单入口多出口的方法体，这种方法体内有多个 `return` 语句用来退出方法体。图 16-2 就是这两者之间的差别。

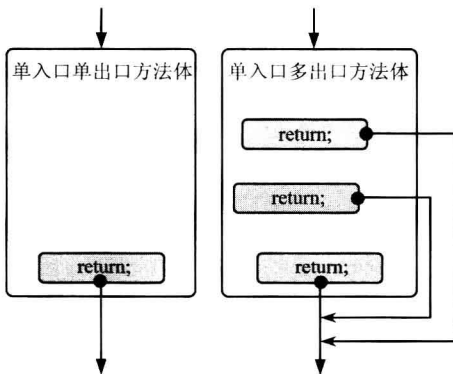


图 16-2 两种方法体的比较

在实践中应该推荐单进口单出口的原则，其好处有：

(1) 单进单出的方法体中，可以在唯一的 `return` 语句前对方法返回值进行统一处理；而对于单进多出的方法体，需要在多处写代码进行处理，很容易导致代码重复。

(2) 单进单出的方法体容易调试，可以在唯一的 `return` 语句前设置断点，这样可以在即将退出方法体时中断并调试代码。而对于单进多出的方法体需要设置多个断点，比较麻烦。

在实践中完全遵守这个原则比较难，也显得很教条。因此，在检查方法参数时，若参数值错误而无法执行功能，则可以使用 `return` 语句退出方法体；在功能性代码中尽量向这个原则靠，但不必死守这个原则。

16.2.3 最小权限原则

C#是基于.NET 框架的，而.NET 框架相对于微软以前的软件开发框架来说，对安全性进行了本质性的提升，此时使用 C#进行软件开发需要注意最小权限申请原则。

所谓最小权限原则是指软件在保证功能能正常执行的情况下，只访问当下所需的信息或资源，而尽量避免访问不需要的信息或资源，应用程序对系统的权限只是刚刚好让程序功能正常运行。

可以这么理解最小权限原则，对于人类社会，绝对的权力导致绝对的腐败。也就是说，一个人权力如果非常大，必然会导致腐败，会占据大量的社会资源而不合理利用，其个人行为会冲击社会的稳定，破坏社会公平。因此需要建立一套社会制度，使得每个人不能拥有太大的权力，权力必须受到制约，这样社会才能和谐发展。

同样，对于计算机的世界，绝对的权力也导致绝对的安全风险。一个程序具有太大的权限，必然会导致系统的不稳定，因为没有什么程序是绝对稳定可靠的，高权限的程序可能有意或无意地占据大量的系统资源而不合理利用，个别程序的错误会导致整个系统的崩溃。因此程序不能拥有过量的权限，只能拥有尽可能少的权限，这样整个系统才能安全稳定。这就是最小权限原则。

如图 16-3 所示，某应用程序只需要连接数据库就能正常运行，但在程序运行中，若程序没有特别声明，系统会将当前用户的所有权限都赋予这个应用程序，不但包括连接数据库权限，还可能包括读取文件和注册表等重要系统资源的权限。此时应用程序不但有访问数据库的通道，同时还保留着访问文件系统和注册表的通道。

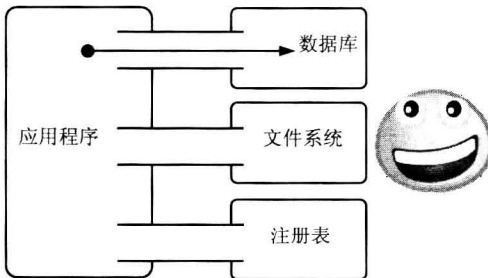


图 16-3 程序运行过程

当应用程序正常运行时，能完成应有的功能，同时不会对用户造成任何安全问题，此时用户没有发现问题，非常满意。

不过现在的软件运行环境的安全性比较复杂，病毒、木马、黑客攻击等安全风险都是必须考虑的，若不考虑则会损害用户的利益，如图 16-4 所示。

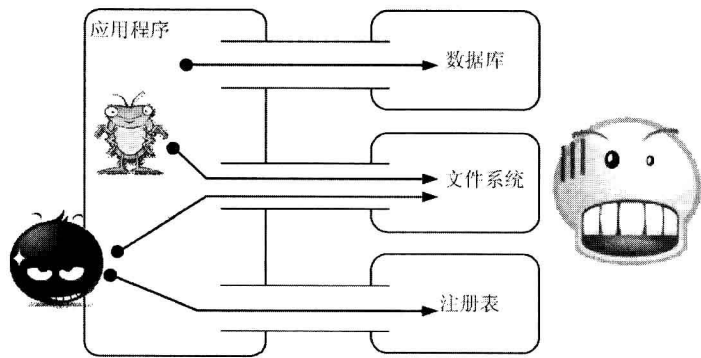


图 16-4 程序运行过程

若应用程序被黑客攻击并被挟持，则黑客会通过应用程序执行不应该执行的功能。例如访问用户系统的文件系统或注册表，读取或写入敏感数据，造成信息安全事故，这会损害用户的利益。

应用程序可能存在 BUG，比如应用程序在开发过程中存在保存调试信息文件的功能，实际运行过程中不应当执行这样的操作，但由于程序错误而意外地执行了这样的操作。此时应用程序会写大量的垃圾文件，浪费用户系统的存储资源，也损害用户的利益。

这时候应用程序是不安全的，存在很大的安全风险。

若开发人员在开发这个应用程序时遵循了最小权限原则，主动让应用程序声明没有访问文件系统和注册表的权限，则此时应用程序的运行如图 16-5 所示。

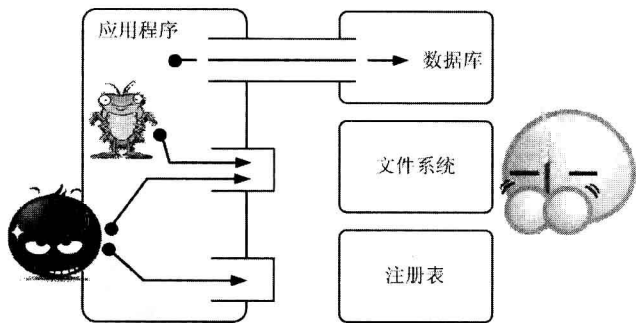


图 16-5 程序运行过程

虽然有黑客劫持了应用程序，或者应用程序本身存在 BUG，使得应用程序试图访问用户的文件系统或注册表，但由于事先声明没有这方面的权限，因此应用程序无法执行这些操作，反而

会导致应用程序错误，提醒用户发生了异常。这样开发人员的深谋远虑保护了用户的利益，用户很庆幸有这样称职的开发人员替他们开发应用系统。

根据上面的分析可以看出，最小权限原则是解决安全风险比较有效的技术手段。能够尽量避免由于黑客攻击或程序错误而导致的安全问题带来的后果，增强系统的安全防御能力。

由于没有权限而导致的拒绝访问的程序错误很容易导致应用程序无条件立即退出，此时用户输入的数据会来不及保存而丢失，这会损害用户的利益。而在开发过程中使用最小权限原则能尽早暴露并解决这个问题进而避免这种情况，保护用户的利益。

在开发实践中遵守最小权限原则，需要开发人员始终保持安全风险意识，认识到未来的应用程序可能面临着各种各样的安全风险，因此需要小心翼翼地开发应用程序，尽量降低应用程序正常运行所需要的权限，主动放弃不必要的权限，精打细算。

例如在读取文件内容时，只需以只读的方式打开文件，而不是以可读写的方式打开文件；若要保存临时文件，则将临时文件保存在临时目录下，而不是保存在应用程序目录或系统目录下；系统配置能保存到系统配置文件中就不要保存到系统注册表中，等等。

现在环境下，信息安全事故频发，有些事故损失还比较大。因此每一个软件开发人员都必须树立起安全意识，充分考虑到各种安全风险，采用各种技术手段来开发出安全的软件。此时最小权限原则就显得比较重要了。

16.2.4 尽早暴露错误原则

开发者应当遵守“尽早暴露错误”的开发原则，这是因为程序错误是客观存在的事实，应当正视它并有效地处理它，而不是简单粗暴地和谐掉。在开发中应当尽早暴露出程序的错误，这有助于发现错误的本质，帮助改善程序质量。若一味地和谐掩盖错误，则错误越拖影响越大，最后不可收拾，程序崩溃。

例如，数据库中有一个名为 Customers 的数据表，其内容如表 16-3 所示。

表 16-3 Customers 数据表

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
1	少室山公司	方证	采购员	东园西甲 30 号	长平
2	擎天航空	雷震子	销售代表	常保阁东 80 号	莫斯科
3	华夏工程	李大禹	市场经理	广发北路 10 号	幽州
4	武当投资	宋青书	物主	临翠大街 80 号	巴伐利亚
5	擎天南京公司	大星星	物主	花园东街 90 号	许安

可以写出以下两种代码来读取并输出其中的数据。

第一种：

```
using (IDbConnection conn = new OleDbConnection())
{
    //连接数据库
```

```
conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
    + System.IO.Path.Combine(Application.StartupPath, "Customers.mdb");
conn.Open();
using (IDbCommand cmd = conn.CreateCommand())
{
    //设置 SQL 语句
    cmd.CommandText = @"
        Select
            CustomerID ,
            CompanyName ,
            ContactName ,
            ContactTitle ,
            Address ,
            City
        From Customers";
    //读取并输出数据
    IDataReader reader = cmd.ExecuteReader();
    if (reader.Read())
    {
        Console.WriteLine("CustomerID =" + reader["CustomerID"]);
        Console.WriteLine("CompanyName =" + reader["CompanyName"]);
        Console.WriteLine("ContactName =" + reader["ContactName"]);
        Console.WriteLine("ContactTitle=" + reader["ContactTitle"]);
        Console.WriteLine("Address    =" + reader["Address"]);
        Console.WriteLine("City      =" + reader["City"]);
    }
    reader.Close();
}
//using
}
//using
```

第二种：

```
using (IDbConnection conn = new OleDbConnection())
{
    //连接数据库
    conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
        + System.IO.Path.Combine(Application.StartupPath, "Customers.mdb");
    conn.Open();
    using (IDbCommand cmd = conn.CreateCommand())
    {
        //设置 SQL 语句
        cmd.CommandText = @"Select * From Customers";
        //读取并输出数据
        IDataReader reader = cmd.ExecuteReader();
        if (reader.Read())
        {
            Console.WriteLine("CustomerID =" + reader["CustomerID"]);
            Console.WriteLine("CompanyName =" + reader["CompanyName"]);
            Console.WriteLine("ContactName =" + reader["ContactName"]);
            Console.WriteLine("ContactTitle=" + reader["ContactTitle"]);
            Console.WriteLine("Address    =" + reader["Address"]);
            Console.WriteLine("City      =" + reader["City"]);
        }
        reader.Close();
    }
}
//using
}
//using
```

这两种代码唯一的区别在于其中使用了不同的 SQL 语句，第一种代码使用的 SQL 语句是：

```
Select
```

```

CustomerID ,
CompanyName ,
ContactName ,
ContactTitle ,
Address ,
City
From Customers

```

而第二种代码使用的 SQL 语句是：

```
Select * From Customers
```

第一种 SQL 语句字符比较多，若开发者比较懒的话，能很容易写出第二种 SQL 语句。推荐使用第一种 SQL 语句，因为第一种 SQL 语句符合尽早暴露程序错误的原则。

如果数据库没有问题，则两种 SQL 语句都能执行，程序都能正确地读取数据，此时第二种写法反而貌似不错。

不过现实中应用程序开发和运行环境都很复杂，比如发生了代码中字段名拼写错误、数据库字段结构发生改变，这些都是需要考虑到问题。

例如数据表 Customers 中的字段 Address 由于某种原因删掉了，于是这两种代码都会发生错误。

对于第一种代码，程序在执行代码“`IDataReader reader = cmd.ExecuteReader();`”时就会报出异常“至少一个参数没有指定值 `OleDbException`”。此时开发者借助 VS.NET 可以很快地确定发生错误的代码，并根据错误提示很容易猜测 SQL 语句写错了。于是开发者进行 SQL 语句与数据库结构的对比，很快就能发现错误的本质，那就是字段 Address 突然没了。

而对于第二种代码，程序顺利地执行了代码“`IDataReader reader = cmd.ExecuteReader();`”，但在执行代码“`Console.WriteLine("Address =" + reader["Address"]);`”时报出异常“`Address IndexOutOfRangeException`”。对于这个错误，开发者会有些迷惑，不知为什么发生错误，因为错误提示信息和数据库联系不大。然后怀疑代码中的“`reader["Address"]`”出现字段名拼写错误，花上一段时间仔细校对确定无误后才会往前继续寻找错误的来源，发现 SQL 语句没有拼写错误。最后才会想到去查数据库结构，绕了半天才发现数据库字段 Address 没了，这才是错误的本质。

在这个例子中，从数据库字段 Address 删掉的那一刻起，系统就存在隐患，而第一种代码能在第一时间由于这个隐患而报出错误，开发者就能立即定位到离隐患最近的代码，得到与隐患密切相关的错误提示信息，也就能非常快地发现问题的本质，从而解决问题。这是一种将错误扼杀在摇篮当中的做法，对开发者对程序都有好处。

而第二种代码很和谐，没能在第一时间让隐患报出错误，让隐患一直默默地影响着程序的运行，最终隐患会报出更大的更让人摸不着头脑的错误，使得开发者需要花费更多的时间和精力来处理这个错误。这是姑息隐患，将错误“养大了出栏再杀”的做法，对开发者对程序都有害处。

在 VB 中有一个写做“`On Error Resume Next`”的语句，号称能和谐掉程序中所有的错误，

让程序不报出任何异常，这是彻彻底底的破坛子破摔、掩耳盗铃的做法，因此这种写法应用得比较少。

开发者要写出一个真正和谐的程序，其过程必然不能简单粗暴地和谐，要尽可能早地让各种隐患报出程序错误；隐患要变成错误，程序不能打击压制，程序要疏通各种隐患“上访”的通道，让开发者更快地发现错误的本质，更快地解决错误。尊重科学规律，正视隐患和错误，正确处理错误而不是打压隐患，程序才能和谐健壮。

面向对象软件开发方法

在商业软件开发中，存在巨大的软件功能模块和编码量，代码量可能有数十万甚至上百万行。但开发项目组人数是有限的，因此业界普遍采用面向对象编程方法，让有限的人来开发和维护巨大的软件项目。本章介绍面向对象编程方法的基本思想。

17.1 发现问题

随着软件行业的发展，客户也在发展，现在的客户会提出越来越复杂的软件功能需求，而软件行业的开发能力是缓慢提升的。由于客户是上帝，他们的需求必须满足，因此很多时候软件开发人员疲于奔命，成天忙着实现客户的各种软件功能需求，用户需求就是软件开发人员的加班之源。于是业界存在日益增长的用户需求复杂度和软件开发人员缓慢提升的开发能力之间的重大矛盾。

17.2 分析问题

客户需求复杂是客观存在的事实，必须满足，因此只能分析软件开发人员的开发能力为什么不足。

考察其他行业，比如汽车制造行业，基本上是客户要求什么就做什么，要多少就做什么，甚至做出的概念产品超出一般客户的想象。这是因为汽车制造行业是全机械化作业，具有巨大的厂房，紧密的流水线和数万名熟练工人，还有很多工业机器人参与其中。由于采用流水线生产模式，汽车制造过程的规则简单高效，生产速度快，能充分地满足客户的数量需求。

而软件开发行业，基本上还是手工作坊模式。目前软件开发的水平还不够高，不能进行流水线式的生产，软件开发人员培养成本高，时间长。因此软件开发能力的提升是缓慢的，而且估计在未来很长时间内这种情况是不会改变的。

表 17-1 是传统的机械化制造业和软件行业的对比表。

表 17-1 机械化制造业与软件行业的对比

比 较 项 目	机械化制造业	软 件 行 业
流水线	有	无
生产机械化	高度机械化	弱
从业人员	工作内容简单，快速批量培训，成本低	工作内容复杂，培养难，时间长，成本高
生产时间	可以三班倒	不能三班倒
信息化	MSI, ERP, 数据仓库，能上的都上了	弱

业界存在一个有意思的情况，软件行业为其他行业提供很多信息化服务，但自身的信息化反而很弱，笔者尚未见过针对软件行业的信息化整体解决方案。

由于软件行业还采用手工作坊式的生产模式，因此客户需求复杂度的增长会导致软件开发工作量的飞速增长，由此造成软件行业的开发能力不足，于是开发人员经常加班。

有些客户需求相当复杂，使得软件开发工作量非常大，从而导致开发人员工作量大，软件开发成本急剧上涨，此时这个软件开发任务是不经济和不可行的。

17.3 解决问题

如何用有限的软件开发能力来应付无限的用户需求呢？固然有项目管理方面的手段，在此只讨论面向对象软件开发方法。

在软件开发中，使用面向对象软件方法是一种放长线钓大鱼的策略，首先要高瞻远瞩，然后精耕细作，最后全面开花完成软件开发任务。面向对象编程方法具有任务分解、过程控制和知识重用三大功能。

17.3.1 任务分解

从策略上看，面向对象软件开发方法是指将一个整体上复杂的软件功能拆分成多个功能简单的软件功能，然后以较低的成本实现各个简单的软件功能，最后进行模块集成形成一个完整而复杂的软件功能来满足用户需求。采用的是分割包围，各个击破的策略。

图 17-1 展示了面向过程编程方法和面向对象编程方法在软件开发工作量上的对比。

在图 17-1 中，阴影部分的面积就是软件开发工作量。面向过程的软件开发方法，试图用一个软件模块来满足客户的所有需求，这是一种鲸吞的方法，当客户需求很复杂时，这个软件模块将非常复杂，难于控制，而且软件开发工作量将非常大。

而采用面向对象的软件开发方法时，开发人员可以将复杂的客户需求拆分成两个较为简单的客户需求，然后逐一开发，最后进行集成实现所需的功能，这是一种蚕食的方法。

在图 17-1 中对于相同的客户需求复杂度，采用面向对象软件开发方法，其开发工作量降低了一半，此处效果的对比是惊人的。因此采用面向对象的软件开发方法，以较低的成本开发客户

需求复杂的商业软件成为可能。

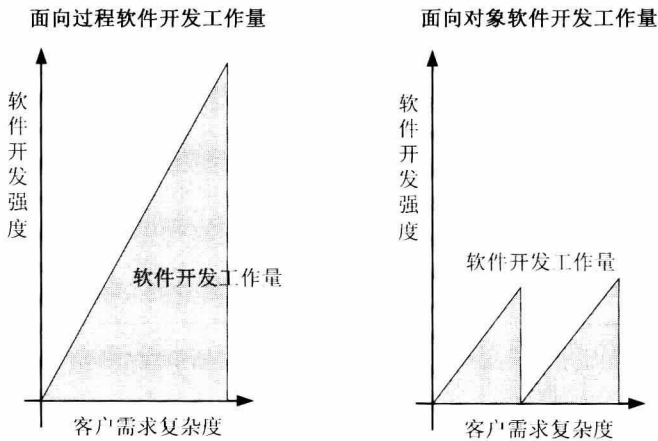


图 17-1 两种方法的对比

对于简单的客户需求，软件开发人员具有足够的能力，采用面向过程的软件开发方法，能鲸吞掉所有的客户需求；但在实际中大量的客户需求是巨大而复杂的，此时鲸吞是不切实际的，而应当采用面向对象的软件开发方法进行蚕食。

面向过程的软件开发方法虽然也有划分软件模块的内容，但其对客户软件功能需求的分割能力不够，不能解决复杂软件分模块的问题；而面向对象编程方法在这方面进行了强化，初步解决了复杂软件模块的分割问题。

17.3.2 过程控制

从软件开发过程看，面向对象编程思想强制添加了一个软件抽象的步骤，迫使软件开发人员深入地了解和分析用户需求，从而增加了软件开发流程的稳定性和正规性。

17.3.3 知识重用

面向对象软件开发方法是来源于生活的，是人们对世界万事万物的认知方法在软件开发领域中的具体应用。

在考察人们认识理解各种事物的过程中，会发现人们会有意无意地将诸多事物进行分类整理和对比。

比如人们经过大量的认识，把汽车抽象理解为至少有 4 个轮子和 2 个车轴的机动车辆，而客车就是专门运输人员的汽车，公交车就是任何人交费就可以乘坐的客车。于是形成了“机动车—汽车—客车—公交车”这 4 种客观事物和由 3 个派生关系构成的关于汽车的系统认识。这个认识方式学习成本低，速度快，知识能重复利用是其最大的优点。

若不用这种分类比较的方法，则汽车就定义为至少有 4 个轮子 2 个车轴的机动车；客车就是

专门运输人员的至少有 4 个轮子和 2 个车轴的机动车；公交车就是任何人交费就可以乘坐的专门用于运输人员的至少有 4 个轮子和 2 个车轴的机动车。对于这两种认知方式可以进行如表 17-2 所示的对比。

表 17-2 两种认知方式的对比

事物	分 类 对 比	不分类对比
汽车	至少有 4 个轮子和 2 个车轴的机动车	至少有 4 个轮子和 2 个车轴的机动车
客车	专门运输人员的汽车	专门运输人员的至少 4 个轮子和 2 个车轴的机动车
公交车	任何人交费就可以乘坐的客车	任何人交费就可以乘坐的专门用于运输人员的至少有 4 个轮子和 2 个车轴的机动车

显而易见，若不采用分类对比的认知方式，则人们认知事物的成本就会大为增加。但事实上，智力正常的人都自觉地采用这种分类对比的认知方法，从而轻易地认知周围所有的事物。

分类对比的认知方法其最大的好处就是知识的重用。由于知识的重用，人们遇到新事物时无须花时间重新彻底地认识，只需要找到和该新事物比较接近的已知事物，然后进行分类对比，就能很快地认识这个新事物，而且这个新认识的事物在未来的认知中也能发挥作用。

分类对比是一种方法策略，可用于对任何事物的理解。上面关于汽车的例子是分类对比方法在认识汽车时的具体应用。类似地，在软件开发过程中也可采用分类对比的方法。

软件是开发人员对客户需求的认知的表述。不理解功能需求就不可能开发出正确的软件。就像汽车一样，用户需求也是客观存在的事物，面对复杂的客户需求，采用分类对比的方式是明智之举。分类对比的认知方法在软件开发领域中的具体应用就是面向对象编程方法。而分类对比中对知识的重用在软件开发领域中就表现为代码的重用。

代码就是知识。开发人员理解了客户需求后就产生了关于用户需求的知识，然后使用多种方式表现出来，可以使用自然语言写出用户需求说明文档，也可以使用计算机编程语言写出代码，因此代码只不过是知识的一种表现方式而已。商业软件开发其本质就是对客户需求的理解，而代码只不过是翻译而已。

表 17-3 展示了分类对比的方法和面向对象软件开发方法的对比。

表 17-3 两种方法的对比

对 比 项 目	分类对比方法	面向对象软件开发方法
应用领域	认识世界上的万事万物	认知客户的软件需求
产出	关于世界的知识	关于客户需求的知识
重用	知识的重用	代码的重用
表现形式	自然语言等	文档，计算机编程语言
目标	认识自然，改造自然	认识客户需求，开发出正确的软件

分类对比是一种哲学，面向对象软件开发是一种科学，而写代码就是一种具体技术。认识到

这些关系，新手们就可以用常见的生活哲学指导软件开发，将生活中的各种知识应用到软件开发过程中，这样进行商业软件开发也就不是很难了。

17.3.4 代码重用

分类对比的认知方式其最大的优势就是知识的重复利用，而面向对象编程思想其最大的优势则是代码的重复利用。代码重用有两大好处，其一就是代码寿命明显变长，能最大限度地保护客户的投资，其二就是能层层积累，降低对新知识的学习量。

软件需要读取文件中的数据，而文件是保存在硬盘上的，因此程序需要精确控制磁头的位置来读取硬盘上的文件数据。读取文件数据是一种非常普遍的使用需求，因此操作系统开发商们针对这种客户需求开发了大量的文件操作程序供软件开发人员使用。此时就产生了操作文件的程序代码的重用。现在业界使用的操作系统中的一些代码有十几年甚至更长的寿命。因此代码的重用能保护业界在系统软件中的投资，使得软件开发人员不必一遍又一遍地编写复杂的读取文件的底层代码。

代码重用能实现软件开发知识的积累。这种积累既包括现成的软件代码、软件模块积累，还包括无形的软件开发技术的积累。

建议毕业生们从现在开始进行代码的积累，代码积累不是代码的简单堆积，它需要修改代码来重用，即以重构来实现重用。代码重构也是一个大话题，在此不讨论了。

面向对象的编程方法就是分类对比的方法在软件开发中的具体实现。

代码就是客户已有的投资结果，商业软件开发人员应当充分考虑到对客户已有投资的保护，在成本允许的情况下旧代码应当尽量保存。

代码是软件的细胞，代码的长寿是软件长寿的基础，也是客户投资价值最大化的基础。

软件的代码分为 3 种，第一种是通用代码，可构造多个通用的标准软件模块，这些标准软件模块不但能用于这个软件，还能广泛地用于其他软件；第二种是特定功能代码，它组成特定软件功能模块，是针对特定的用户需求而开发的功能代码，只能用于这个软件；第三种就是黏合剂代码，就是将标准软件模块和特定软件模块进行整合，从而将多个软件模块拼凑成一个软件。

很显然，对于一个开发多个应用系统的开发者来说，它所掌握的通用软件模块越多，应用系统的平均开发工作量就越少，成本就越低，越能适应价格战。

面向对象编程思想可适应当前软件开发规模越来越大，复杂度越来越高的情况。软件开发的工作量随着软件规模的增加而呈非线性增长，软件规模越大，软件开发工作量增长的速度越快。使用传统的面向过程的编程方法，开发人员的工作量和软件开发工作量是成正比的，因此当软件规模很大时，开发人员的工作量将非常巨大，从而导致软件开发成本急剧上升，这对于商业软件开发是不可行的。软件客户的复杂需求是客观存在的，是需要满足的，因此业界开始使用面向对象的编程方法来降低软件开发成本。

17.4 面向对象开发

17.4.1 封装

1. 接口信息爆炸

在人类社会中，公平、公开、公正是社会稳定和谐的基础，这是因为人的思想太灵活了，不公开就是封闭（封装），必然导致暗箱操作，绝对的权力导致绝对的腐败。

但在机器的世界中，机器永远是按部就班、秉公执法的，因此无所谓公平、公开、公正，只需要把程序设置好就可以了，也就谈不上封装了。

但对于软件开发人员来说，是否封装就是一件要认真考虑的事了。

现在软件非常复杂，一个软件系统中有很多类型，每个类型包含很多个字段、属性和方法，若这些类型及其成员都是公开的，则能供其他程序模块调用的接口将有成千上万，会形成接口信息爆炸。

开发人员需要在正确的时机调用正确的接口，面对这么多的编程接口很容易不知所措，需要花很大的精力记忆这些接口的用法，而且在千挑万选中可能一不留神调用了错误的接口而导致错误的结果。

开发人员在开发软件模块时应进行有意识的封装，尽量减少软件模块对外接口数量。对于类来说，就是减少其公开的成员个数。当软件模块的接口数量少时，开发人员就能方便记忆和正确地调用了，就能避免接口信息爆炸。

2. 安全

软件需要处理各种错误的输入数据，不能假设所有的数据都是正确的，此时需要进行封装来过滤掉一些错误的数据，尽早发现问题。

比如 `PeoleClass` 类型的 `Name` 属性，理论上应该设置为“张三”、“李四”等文本，长度不超过 20 个字符，而且数据库中保存姓名的字段也限制为 20 个字符长度。但实际上程序很可能设置超过 20 个字符的字符串值到 `PeoleClass` 类型的 `Name` 属性。此时保存到数据库中必然会导致数据过长的数据库更新错误。这就是由于程序没有尽早检查错误的数据而导致问题越拖越久，最后严重影响系统的稳定性。

当使用公开字段来设置数据时，开发人员没有任何办法来第一时间检查字段值是否正确。但采用封装，就可以在 `Name` 属性中添加一些代码来第一时间检查属性值是否安全。比如可以使用以下代码来编写 `Name` 属性。

```
private string _Name = null;
public string Name
{
```

```
get
{
    return _Name;
}
set
{
    if ( value != null && value.Length > 20)
    {
        throw new Exception("姓名不得超过 20 个字符");
    }
    _Name = value;
}
}
```

这段代码使程序能在第一时间检查出输入的数据是否正确，也就能够在第一时间暴露出问题。能尽早地解决错误是开发健壮安全的软件的基础。一个软件，若不能及时发现有问题的数据，那么一切稳定都是浮云。

同仁堂药店有个古训“炮制虽繁必不敢省人工，品味虽贵必不敢减物力”。做软件也一样，开发优秀的软件固然需要优秀的软件架构，但也需要优秀的基本功，封装就是软件开发的基本功，做好基本功，优秀的软件也就有了扎实的基础。

17.4.2 继承

知识的重用在面向对象开发中表现为继承。所谓继承就是指在原有类型上派生出新的类型，而且新的类型将继承基础类型的功能。

很多软件开发技术，比如继承、重构等，其内容都可以追溯到知识重用这个朴素的智慧上。

第 18 章

团队开发管理

现代的商业软件开发基本上都是团队开发，有多个人在同项目中进行设计、开发、调试和维护。多人操作必然涉及分工合作。

18.1 项目管理

项目管理是一个软件企业日常经营管理中最为重要的部分，拥有很强的软件项目管理执行力是一个优秀的软件企业的必备条件。

在一个管理比较规范的软件企业中，大多采用比较严格的软件项目管理的基本过程。一个软件项目管理中会涉及的部门和人员如图 18-1 所示。

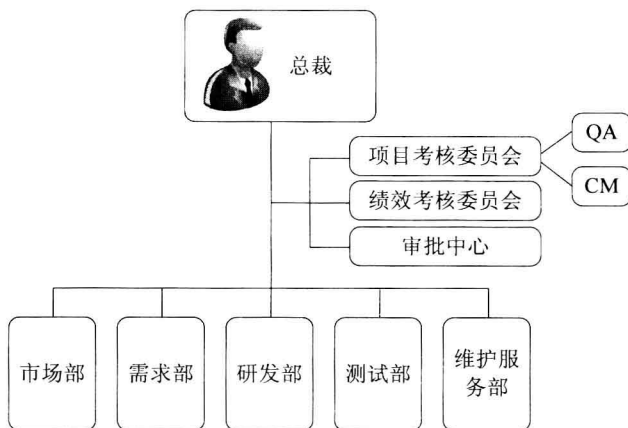


图 18-1 组织架构

在这个和软件项目管理相关的组织架构中，包括的主要角色有：

(1) 总裁：为公司的最高领导，在很多项目管理过程中需要其参与，并最终签名确认。

(2) 项目考核委员会：对项目经理所进行的软件项目管理活动进行考核，包括 QA 和 CM。

(3) QA：质量控制员，负责对项目管理中产生的源代码和文档进行规范性检查，但不检查内容的正确性。

(4) CM: 配置管理员, 负责管理公司的源代码和服务器, 包括项目文档目录、相关人员对服务器的访问权限等。

(5) 绩效考核委员会: 负责对所有人员的工作内容、工作绩效进行考核和评判。

(6) 审批中心: 负责项目费用的审查批准工作。

(7) 市场部: 能提供软件项目的合同, 是一个软件项目的发起者。

(8) 需求部: 进行软件需求调研、分析工作。

(9) 研发部: 根据软件合同及客户需求进行软件研发。

(10) 测试部: 对研发部开发的软件进行测试。

(11) 维护服务部: 软件项目完成后, 从研发部接手软件项目, 进行售后服务。

一个完整的软件研发项目分为项目启动、需求开发、项目计划、系统设计、开发实施、系统测试、发布部署、试用验收、项目结项、项目移交、项目管理等步骤。每个步骤都会产生很多书面文档。

18.1.1 项目启动

销售人员接到单子, 签订合同, 然后就开始启动一个软件项目, 这是项目管理中的第一个阶段。但以项目启动作为软件项目管理周期的第一个阶段是不够的, 建议向前延伸扩展到市场销售环节。

在很多公司的利益架构中, 销售人员的利益和软件开发人员的利益是分开的。销售人员以软件销售合同为纲, 为拿单子大打价格战, 结果有时会签订以较低合同金额来实现较多内容的软件项目合同, 性价比不好, 不符合公司利益的最大化, 而且也对后面的项目管理带来困难。有些公司的销售人员拿到单子签订合同就能获得所有的收益, 然后就没有动力管项目了。因此技术人员应当提前介入软件销售环节, 技术人员向销售人员提供技术方面的参考, 特别是提出与软件研发成本相关的专业建议, 避免签订注定要亏本的合作。

在项目管理中有很多和客户相关的管理, 比如需求变更、客户不够配合工作等。由于销售人员具有很好的客户关系和沟通技巧, 处理起来不算难, 有能力维护公司的利益; 而技术人员不擅长这些, 处理起来比较费劲, 很难做到公司利益的最大化。

没有利益因素驱动, 销售人员一般不会帮助技术人员解决这些与客户相关的问题, 很多时候他们会撒手不管, 让技术人员很吃力地处理这些问题, 这样增加了项目开发成本, 最终不符合公司利益的最大化。

因此应该有一种机制让技术人员向前延伸, 销售人员向后延伸, 这样能发挥各自的特长, 共同维护公司的利益。

18.1.2 需求开发

项目立项后, 就进入软件需求开发阶段, 该阶段的具体流程如图 18-2 所示。

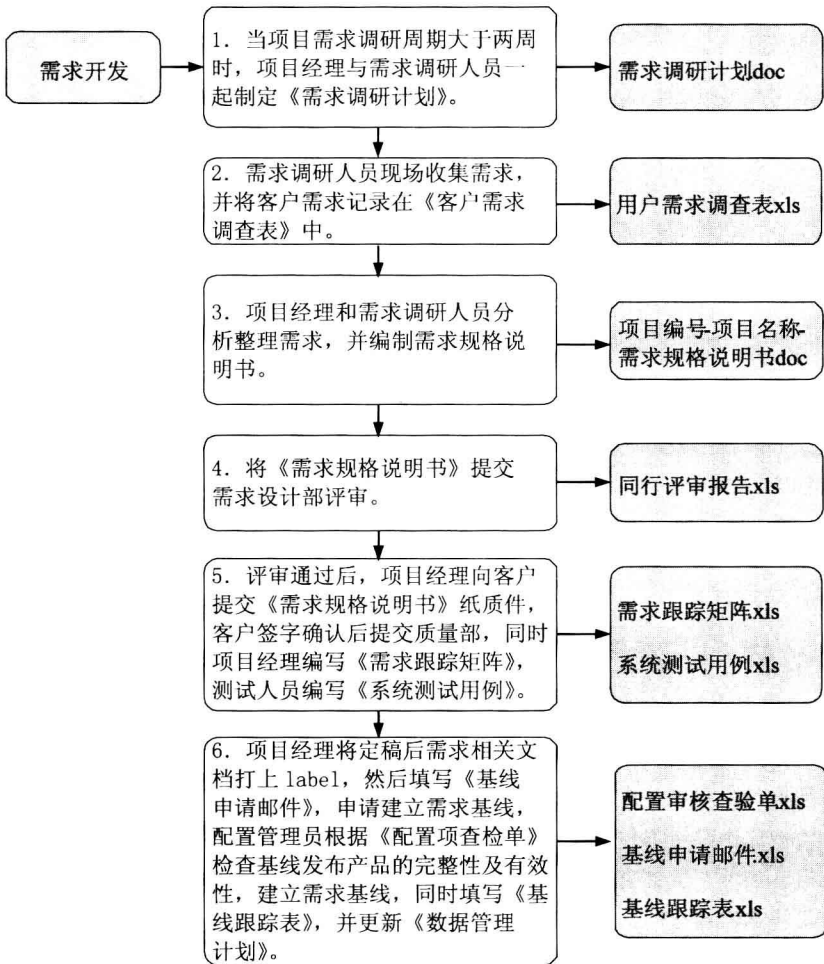


图 18-2 软件需求开发流程

需求是软件项目管理的基础，这一步没做好，以后的工作做得再好，也会导致“我猜中了前头，可是我猜不着这结局”的悲剧。

不过现在需求变更得太厉害了，必须以灵活的项目管理手段和技术手段来缓解这个问题。

18.1.3 项目计划

项目计划就是为整个项目的研发和部署制定一个总体的计划，其流程如图 18-3 所示。

18.1.4 系统设计

系统设计就是为系统进行设计，其流程如图 18-4 所示。

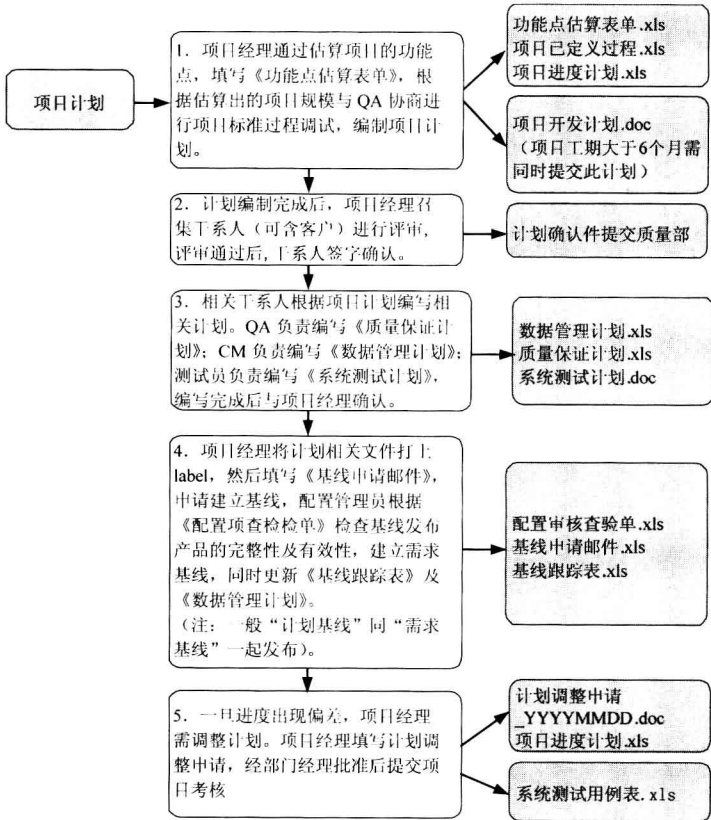


图 18-3 项目计划流程

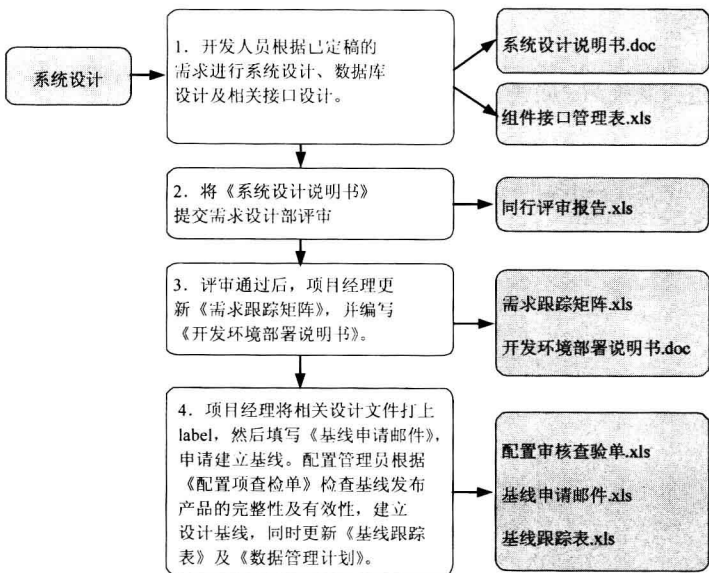


图 18-4 系统设计流程

18.1.5 开发实施

一个软件项目经过系统设计后，就进入了开发实施阶段，该阶段的执行流程如图 18-5 所示。

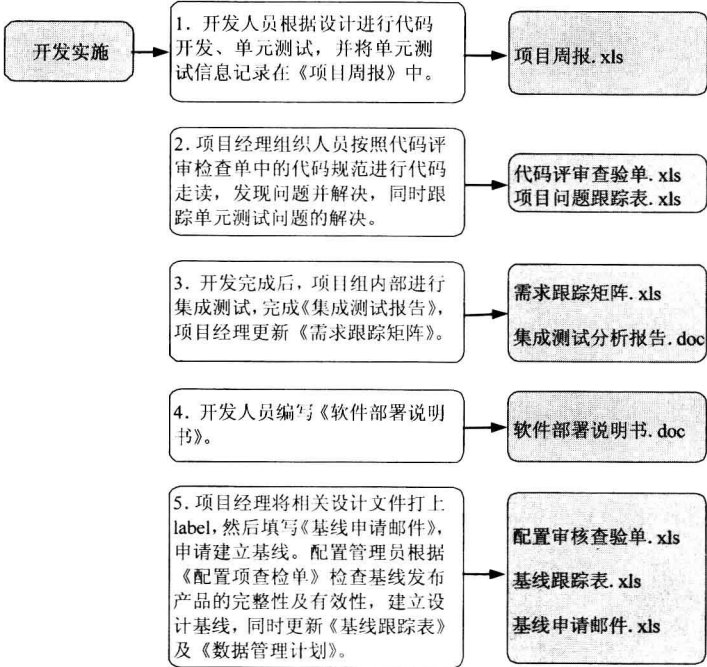


图 18-5 开发执行流程

18.1.6 系统测试

系统开发后进入系统测试阶段，该阶段的流程如图 18-6 所示。

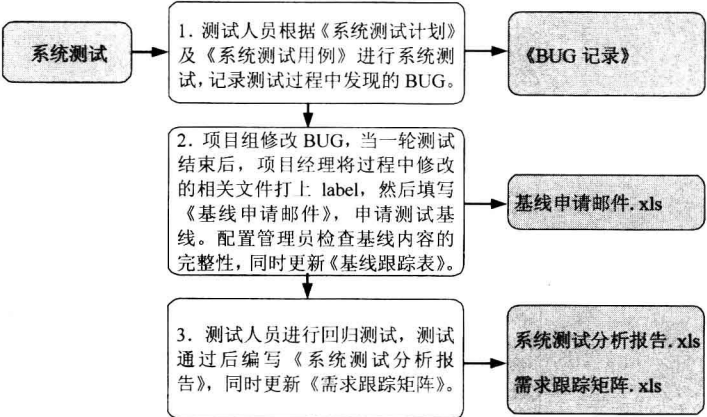


图 18-6 系统测试流程

18.1.7 发布部署

系统测试通过后即进入发布部署阶段，该阶段的流程如图 18-7 所示。

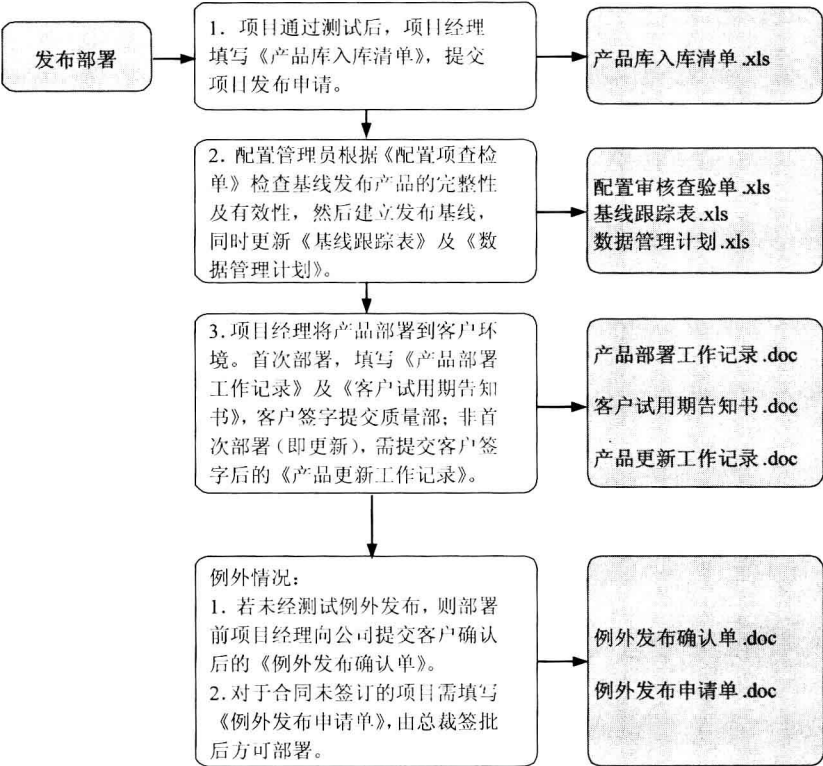


图 18-7 发布部署流程

注意，例外发布需要得到公司总裁的签字批准，而且例外发布也不太符合规范，因此尽量避免。

18.1.8 试用验收

系统在客户处发布部署后，就需要请用户试用，然后验收项目。这个阶段的流程如图 18-8 所示。

18.1.9 项目结项

系统经过用户试用和验收后，该项目的开发环节就结束了，此时进入项目结项阶段，该阶段的流程如图 18-9 所示。

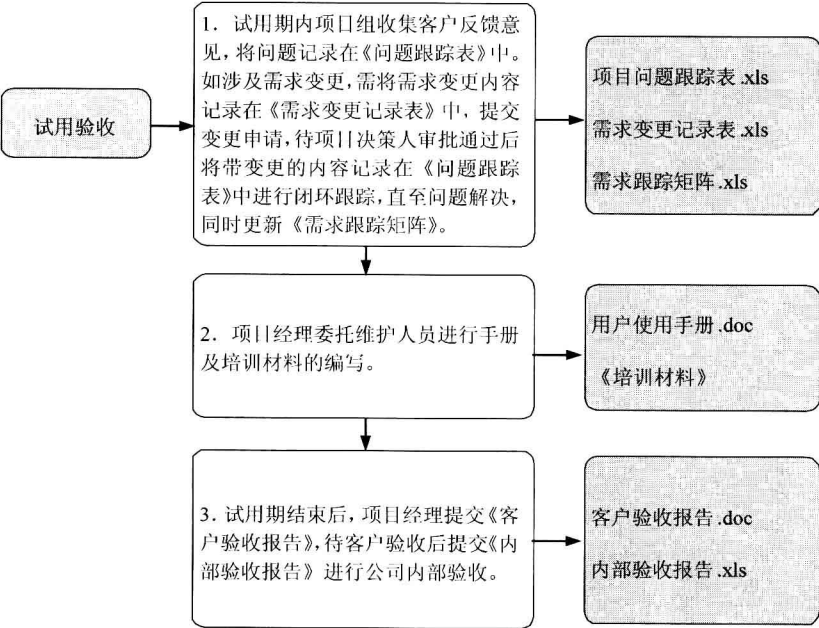


图 18-8 试用验收流程

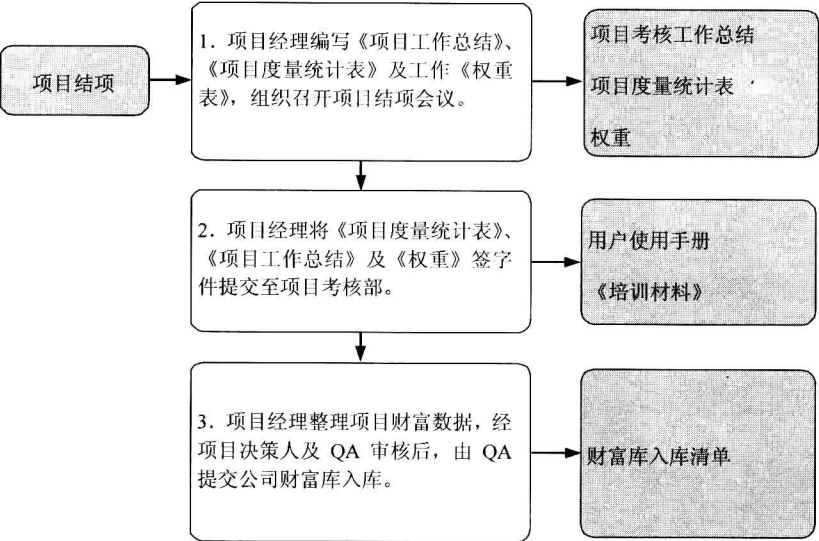
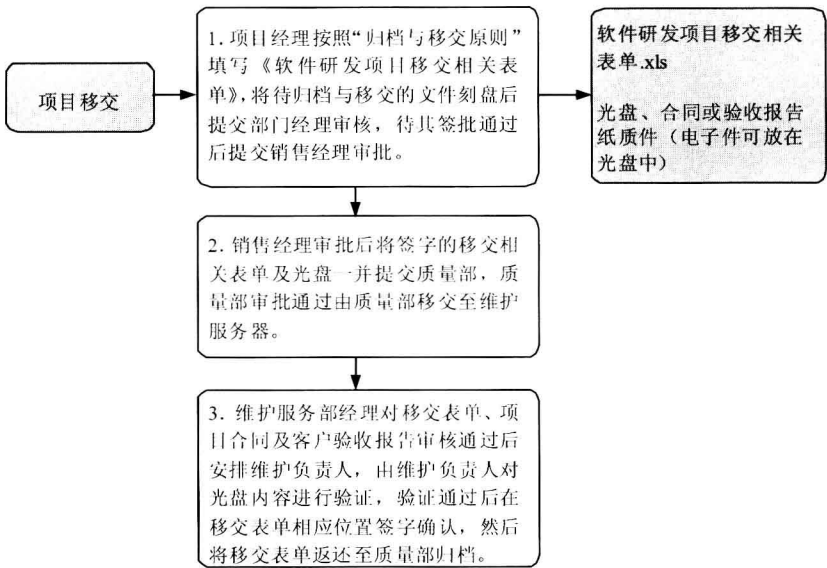


图 18-9 项目结项流程

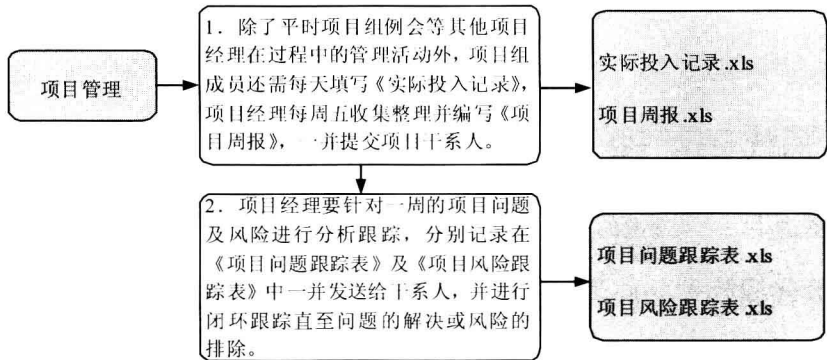
18.1.10 项目移交

项目结束后，研发部门将项目移交给维护服务部门，项目进入售后服务阶段，这个过程如图 18-10 所示。



18.1.11 项目管理

项目经理负责整个项目研发管理，其职责如图 18-11 所示。



18.1.12 QA

QA，也就是质量管理员，其工作职责如图 18-12 所示。

18.1.13 CM

CM，就是配置管理员，其职责如图 18-13 所示。

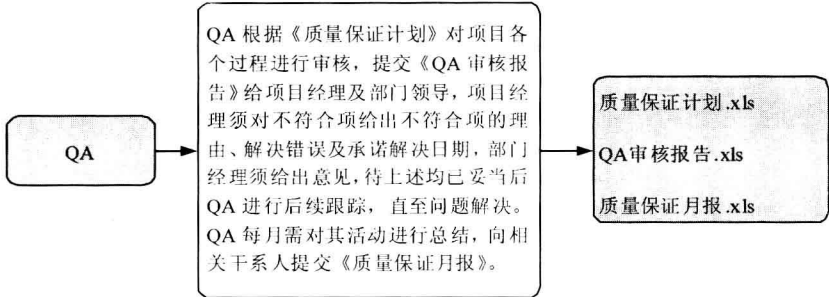


图 18-12 QA 工作职责

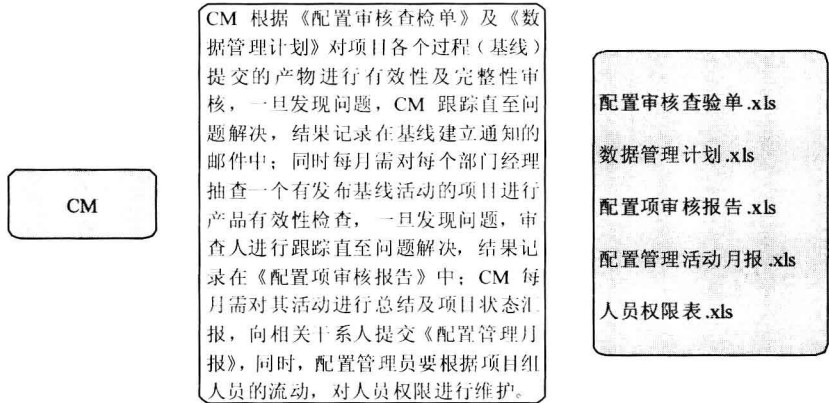


图 18-13 CM 工作职责

18.2 源代码管理

在项目管理中，非常重要的一项工作就是源代码管理。

18.2.1 源代码管理的原理

在团队开发中，可能是多个人共同编写一个文档文件，也可能是多个人同时编写一个 C# 解决方案中的源代码文件。这就存在分工与合作的问题。

使用 VS.NET 编译程序时，需要读取本地计算机目录中的源代码文件进行编译。由于多人开发肯定是在多台计算机上进行的，因此存在多台计算机上的源代码文件同步的问题。

比如在一个 C# 项目中有一个 CS 源代码文件，开发人员甲在计算机 A 修改了这个 CS 文件，添加了新的类型和方法，而开发人员乙在计算机 B 上进行开发工作。此时需要从计算机 A 复制所有修改的源代码文件到计算机 B 中，这样才能进行开发和编译。这个手工操作过程如图 18-14 所示。

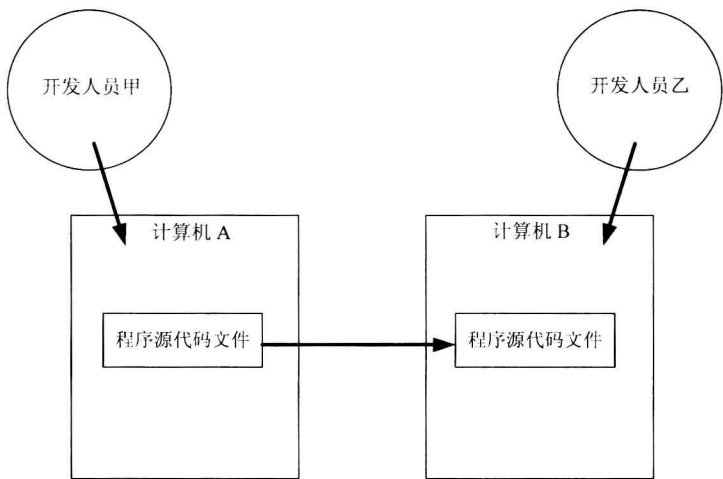


图 18-14 手工操作过程

手工操作过程比较烦琐，很容易出错。若项目中有大量的程序文件，而且有 3 个甚至更多的开发人员同时修改这个项目的源代码，则这个过程不可控，此时就需要源代码管理了。

图 18-15 就是源代码管理的原理图。

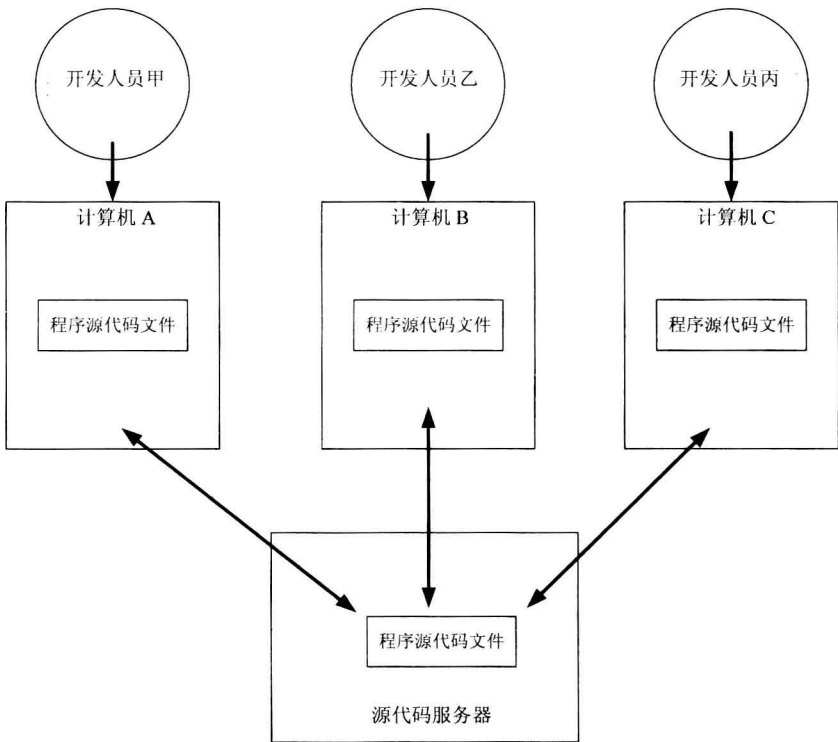


图 18-15 源代码管理原理图

在该原理图中，有一个源代码服务器，所有用于开发的计算机都连接到这个服务器。程序源代码文件不是在各台用于开发的计算机中来回复制的，而是将各自最新的文件都复制到源代码服务器中，也都从源代码服务器中下载新版本的程序源代码文件。这样就避免了难以控制的文件复制操作，确保不因错误的文件覆盖而导致数据丢失。

图 18-16 是源代码管理过程的示意图。

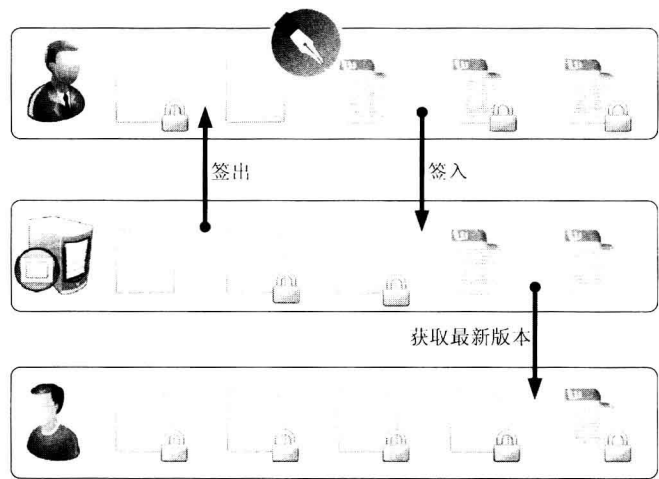


图 18-16 源代码管理过程示意图

该示意图说明了源代码管理过程中的 5 个步骤。

(1) 初始状态。源代码服务器上、各个开发人员的计算机中都有一个程序文件，但在开发人员的计算机中，文件是标记为只读的，不能修改。

(2) 签出。开发人员甲想要修改文件内容，于是执行签出操作，对服务器上的文件加锁，且将本地文件解锁。此时本地文件的属性是可读写的，而服务器上的文件被加锁，使得其他开发人员不能对这个文件进行签出操作。

(3) 修改文件。开发人员甲修改了本地文件内容，但没有同步更新服务器中的文件内容。

(4) 签入。开发人员甲对修改后的文件执行签入操作，将文件内容上传到服务器，从而修改了服务器上的文件内容；然后对本地文件加锁，设置为只读，对服务器上的文件解锁；最后通知其他开发者服务器上的文件是最新版本的。

(5) 获取最新版本。其他开发人员对服务器上的文件执行获取最新版本的操作。将服务器中的文件内容覆盖在本地文件内容上，但本地文件仍然是加锁的、只读的，不能修改内容。

在这个过程中，使用签入、签出操作来设置本地文件和服务器文件的锁定状态，能避免多人同时编辑文件内容的冲突；让各个开发人员通过获取最新版本的操作来进行安全的文件分发复制。这样源代码管理就能保证一个多人的团队协同工作，共同维护一个应用系统的源代码文件。因此源代码管理是团队开发的基础。

18.2.2 VSS 源代码管理软件


在软件开发业界流行着很多源代码管理软件，比如 VSS 和 SVN 等。下面只介绍使用 VSS 源代码管理软件来管理软件的源代码和文档的方法。

VSS 全称为 Visual Source Safe，是微软的 Visual Studio 软件开发套件的一部分，它是专门管理软件开发过程中的源代码和文档文件的。本文使用的是 Visual Source Safe 2005 版本。

VSS 是一种 C/S 程序，是基于局域网文件及目录共享机制的，也提供 Web 插件。

VSS 有两种使用方法，一种是独立使用，另一种是集成在 VS.NET 中使用。

18.2.3 使用 VSS 客户端软件

VSS 默认的安装目录是“C:\Program Files\Microsoft Visual SourceSafe”，它的程序图标是“”。安装好 VSS 后，用户可以展开 Windows 开始菜单，然后展开“Microsoft Visual SourceSafe”，单击“Microsoft Visual SourceSafe”即可独立运行 VSS 客户端软件。

1. 系统登录

启动 VSS 客户端后系统显示出用户登录对话框，如图 18-17 所示。

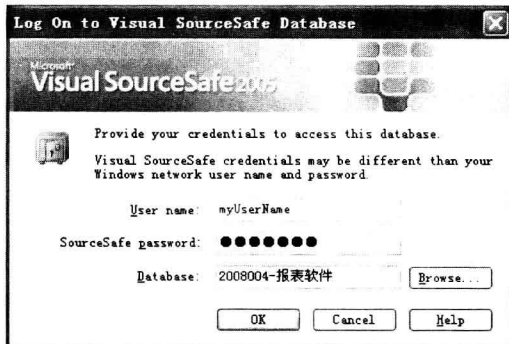


图 18-17 用户登录对话框

在该对话框中输入 VSS 管理员分配好的用户名和密码，然后选择正确的 VSS 数据库即可登录系统。

一个开发组织中可能有多个 VSS 文件服务器，因此可以在 VSS 登录对话框中单击“Browse”按钮来显示如图 18-18 所示的 VSS 数据库选择对话框。

在该对话框中，用户可以选择一个以前配置好的 VSS 数据库，单击“Open”按钮连接该数据库。

若想连接一个新的 VSS 数据库，可以单击“Add”按钮，打开添加 VSS 数据库对话框，如图 18-19 所示。该对话框是一个向导对话框，需要按向导提示进行操作。

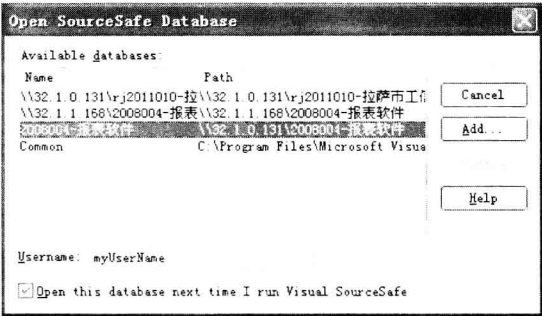


图 18-18 VSS 数据库选择对话框



图 18-19 添加 VSS 数据库对话框（向导 1）

在图 18-19 中单击“下一步”按钮，弹出如图 18-20 所示的对话框。

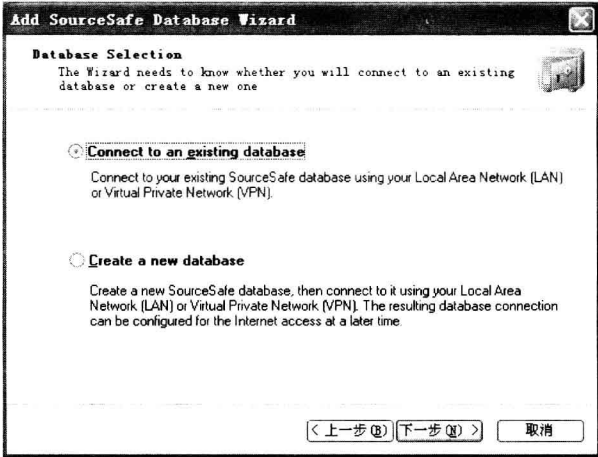


图 18-20 添加 VSS 数据库对话框（向导 2）

在该对话框中，第一个选项是连接现有的数据库，第二个选项是创建一个新的数据库。一般情况下，开发组织中已经有管理员把 VSS 数据库建好了，因此选择“Connect to an existing database”选项。

单击“下一步”按钮，弹出如图 18-21 所示的对话框。

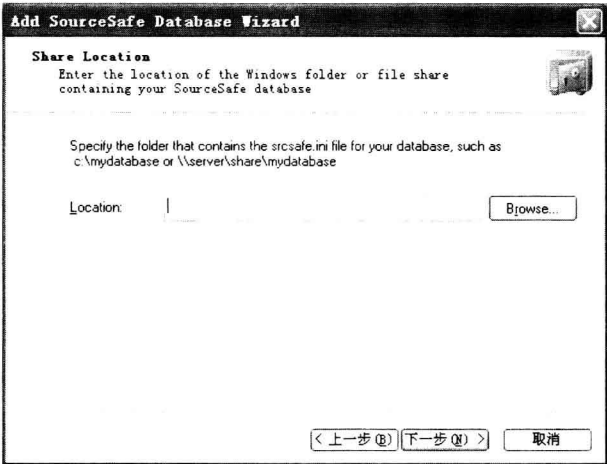


图 18-21 添加 VSS 数据库对话框（向导 3）

在该对话框中，可以在 Location 文本框中输入 VSS 文件目录地址，也可以单击“Browse”按钮，通过浏览选择一个文件目录。这个文件目录通常是网络文件共享目录，例如“\\32.1.0.131\XX 管理系统”，而且这个目录下必须包含一个名为“srcsafe.ini”的文件。

当成功地选择了一个存在的 VSS 文件共享目录后，单击“下一步”按钮，弹出如图 18-22 所示的对话框，此时用户可以为这个数据库取个名字。

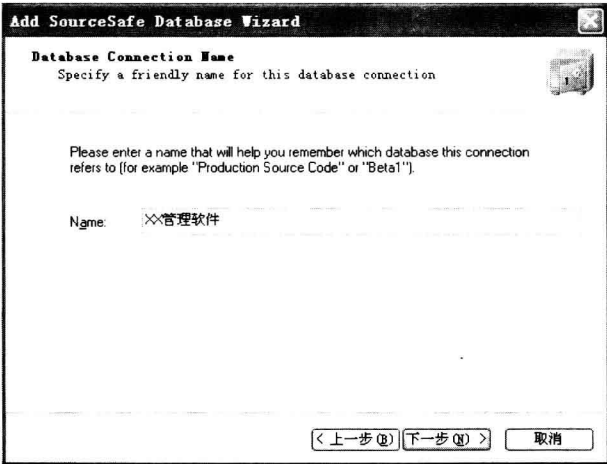


图 18-22 添加 VSS 数据库对话框（向导 4）

单击“下一步”按钮，完成操作，此时的对话框如图 18-23 所示。用户可以单击“完成”按钮完成连接 VSS 数据库的操作。



图 18-23 完成操作

通过以上过程可以知道，登录 VSS 系统需要通过两层安全系统，一个是 VSS 管理员设置的用户名和密码系统，另一个是 VSS 文件服务器管理员对共享目录设置的用户名和密码系统，因此在使用 VSS 前还得连接文件共享目录。

2. VSS 主界面

成功地登录 VSS 数据库后，系统就能显示 VSS 管理器的主界面，如图 18-24 所示。



图 18-24 VSS 管理器主界面

该主界面类似 Windows 资源管理器的主界面，在这个主界面中，上面是菜单和工具条，左边是项目目录结构，右边是当前目录的文件列表，下面是输出文本框和状态栏。

需要特别说明的是，右边的文件视图列表中没有列出子目录。这个列表是可以多选，但不能像 Windows 资源管理器那样用鼠标拖曳多选，而必须用鼠标单击操作结合按下 Shift 键或 Ctrl 键的方式来选择多个文件。

3. 标准文件目录结构

VSS 管理器主界面的左边列出了源代码管理文件目录结构。在项目管理中，若软件项目已经完成立项，则 CM 会为其建立标准的文件目录结构，并为项目相关人员分配权限。该文件目录结构如表 18-1 所示。

表 18-1 文件目录结构

项 目 名 称	说 明
01 开发库	开发过程使用的目录
01 配置管理	开发者提交给 VSS 管理员的入库清单 Excel 文件
02 客户提供的资料	客户提出的一些原始资料
03 项目前期资料	项目前期的一些资料，例如构思，可行性分析报告等
04 项目计划与过程管理	项目的开发计划，相关说明
01 问题跟踪表	
02 日志周报	每周工作小结文档
03 里程碑	
05 同行评审&会议纪要	
06 客户需求及变更需求	
01 需求跟踪矩阵	
02 美工页面	美工设计的图片、图标等
07 系统设计	
01 开发环境部署记录	项目部署的相关记录
08 代码	保存所有的软件程序源代码
09 数据库	开发过程使用的数据库文件，相关的 SQL 脚本文件等
10 测试部署及 BUG 跟踪	测试人员的测试报表等相关文件
11 用户手册	用户手册及相关文件
12 项目维护工作记录	项目维护工作记录及相关文件
13 其他	
02 管理库（受控库）	VSS 管理员使用的库,开发人员不用
03 基线库（受控库）	VSS 管理员使用的库,开发人员不用
04 产品库	VSS 管理员使用的库,开发人员不用

这是一个符合规范的 VSS 源代码管理框架性目录结构，任何开发人员向 VSS 文件服务器上

从毕业生到程序员：使用 C# 开发商业软件

传文件时必须遵守这个目录结构，分门别类，服从管理。

4. 连接 VSS 数据库

VSS 客户端软件只能连接一个数据库，因此在使用 VSS 软件的过程中，有时需要连接其他项目的 VSS 数据库。此时单击软件主菜单项目“File→Open SourceSafe Database”，会显示如图 18-25 所示的 VSS 数据库选中对话框，可以连接到另外一个 VSS 数据库。

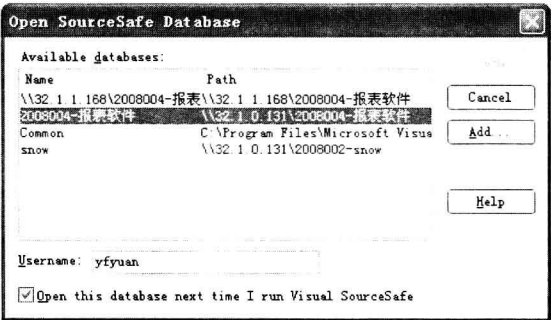


图 18-25 VSS 数据库选中对话框

该对话框中的列表列出了所有以前曾经连接过的 VSS 数据库。若要连接新的数据库可以单击“Add”按钮，在弹出的对话框中可以连接新的数据库。

5. 维护文件目录信息

原则上，配置管理员建立的文件目录框架是不能修改的，但项目组成员可以根据需要在这个框架目录上添加或删除自己的文件和目录结构。

(1) 添加目录

单击主菜单项目“File→Create Project”，会显示如图 18-26 所示的新增项目对话框。

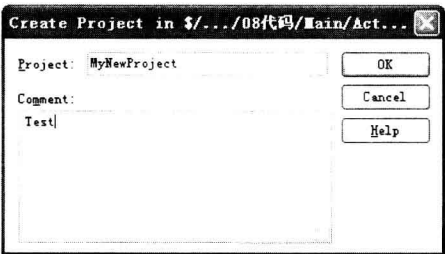


图 18-26 新增项目对话框

在该对话框中输入目录名和说明，单击“OK”按钮，即可在 VSS 项目目录树状列表的当前节点下新增一个子项目。

也可以用鼠标右击目录节点，弹出快捷菜单，单击“Create Project”菜单项目打开该对话框

来添加新目录。

(2) 添加文件

单击主菜单项目“主菜单 File→Add Files”，会显示如图 18-27 所示的文件选择对话框。



图 18-27 文件选择对话框

在该对话框中用户可以选择多个文件，这样就能将多个本地文件添加到 VSS 目录树中当前选中的目录下。

在 VSS 中只能通过添加文件的方式将计算机中已经存在的文件添加到 VSS 目录结构中，不能在 VSS 客户端软件中创建文件。

(3) 删除对象

单击主菜单项目“File→Delete”，即可删除文件或目录。若程序界面中当前获得输入焦点的控件为左边的目录树状列表控件，则会删除当前目录，包括其所有的子目录和文件。若获得输入焦点的控件为右边的文件视图列表控件，则会删除该列表中当前选中的文件。

在删除文件或目录前，系统会显示如图 18-28 所示的对话框来让用户确认操作。

删除后的文件或目录无法恢复，因此需要慎重操作。

在如图 18-29 所示的工具条上有一个“删除”按钮，用户也可以单击工具条上的删除按钮来完成操作，还可以使用快捷键“Delete”来执行操作。

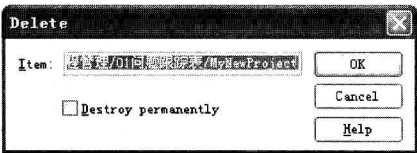


图 18-28 确认对话框

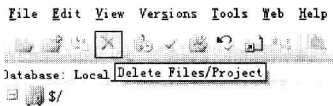


图 18-29 工具条

(4) 恢复文件

当错误地删除文件或目录后，可以还原被删除的对象。具体做法是在目录树状列表中用鼠标右击某个目录，弹出快捷菜单，单击最下面的“Property”菜单项目，弹出该目录的属性对话框，切换到“Deleted Items”选项卡，如图 18-30 所示。

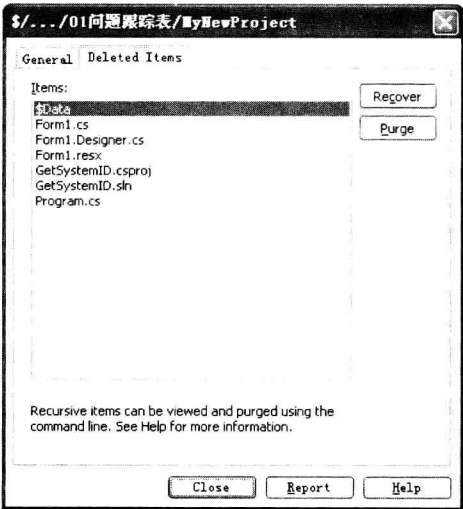


图 18-30 “Deleted Items” 选项卡

在该选项卡的列表中列出了所有被删除的文件和子目录，其中子目录是以字符“\$”开头的。用户可以在这个列表中选择要处理的文件或子目录，然后单击“Recover”按钮即可恢复选中的对象；若用户单击“Purge”按钮则清除被选中的对象，被清除的对象是无法恢复的。一般情况下，CM 不会为普通项目组成员分配这个权限。

(5) 修改名称

用户单击菜单项目“File→Rename”，可以修改文件或目录的名称。或者用鼠标右击文件或目录项目，弹出快捷菜单，在其中单击“Rename”菜单项目也可以修改文件或目录名称。

6. 版本控制

VSS 客户端软件有文件版本控制功能，下面进行具体介绍。

(1) 设置工作目录

所谓工作目录就是本地计算机系统中某个已经存在的文件目录，它与 VSS 中的某个目录存在映射关系。从 VSS 目录中下载的文件将放到这个工作目录中，而上传文件时是将工作目录中的文件上传到 VSS 目录下。

VSS 中每一个目录都可以设置工作目录，VSS 中的子目录默认情况下都映射到工作目录下的同名子目录中，使得 VSS 目录结构和本地系统目录结构保持映射关系。因此在实践中，推荐设置某个级别比较低的 VSS 目录为工作目录，开发人员就能根据 VSS 目录的子孙 VSS 目录，在

工作目录下建立子孙文件目录来实现自动映射关系。

在 VSS 客户端中设置 VSS 目录的工作目录的操作也很简单，就是单击菜单项目“File→Set Working Folder”，会显示如图 18-31 所示的对话框。

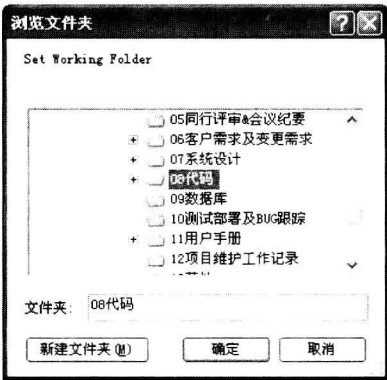


图 18-31 “浏览文件夹”对话框

在该对话框中，用户选择某个本地已经存在的文件目录，然后单击“确定”按钮就能将指定的文件目录作为当前 VSS 目录的工作目录。

在 VSS 目录树状列表中显示的快捷菜单也有这样的功能菜单项目。

(2) 签出

签出操作（Check Out）有两个工作内容：

(1) 将 VSS 目录中的指定文件内容下载到工作目录中，设置本地文件属性为可读写，这样用户就可以编辑本地文件内容。

(2) 标记所有已经签出的 VSS 文件为只读，并加签出操作锁定，这样其他用户只能看 VSS 目录中的文件内容而不能修改它，也不能对它进行签出操作。

在 VSS 客户端中可以对某个目录或多个文件进行签出操作。

- 签出目录

签出目录操作是指签出目录下的 VSS 文件，而不是签出 VSS 目录本身。在 VSS 管理器主界面左边的 VSS 目录结构树状列表中，用鼠标右击弹出快捷菜单，单击快捷菜单中的“Check out”菜单项目，会显示如图 18-32 所示的对话框。

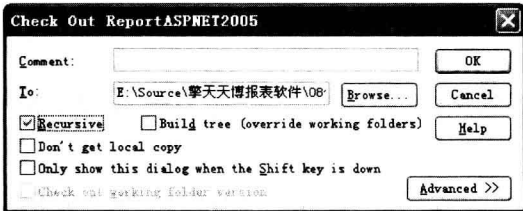


图 18-32 签出目录对话框

在该对话框中，签出路径默认为该 VSS 目录的工作目录，在这个对话框中也可以修改为其他已经存在的本地文件目录。

Recursive 复选框用于指示是否进行文件目录递归签出，若勾选该复选框，则客户端软件会不受层次限制地遍历这个 VSS 目录及其所有的子孙目录，将子孙目录下的所有文件都进行签出操作。

Build tree 复选框用于指示是否创建本地目录结构，若勾选该复选框，则客户端软件会根据需要在工作目录下创建一些子目录，使得 VSS 目录结构和工作目录下的目录结构保持映射关系。但这个操作不会删除已经存在的本地文件目录。

- 签出文件

在 VSS 管理器主界面右边的 VSS 文件视图列表中，用鼠标右击弹出快捷菜单，单击快捷菜单中的“Check out”菜单项目，会显示如图 18-33 所示的对话框。

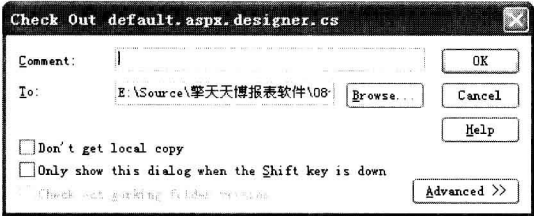


图 18-33 签出文件对话框

该对话框中没有 **Recursive** 复选框和 **Build tree** 复选框。单击“OK”按钮即可签出选中的文件。

文件被签出后，其图标会发生变化，如图 18-34 所示。

Name	User	Date-Time	Check Out Folder
Web.config		11-06-09 11:14	
ReportASPNET2005.c		11-06-28 13:52	
ReportASPNET2005.c		11-06-30 17:12	
Menu.htm		10-10-03 8:49	
Main.htm		09-05-31 10:18	
Global.aspx.cs	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den
Global.aspx	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den
default.aspx.designer.cs	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den
default.aspx.cs	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den
default.aspx	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den
copylist.txt	袁永福	11-08-02 16:07	E:\Source\擎天天博报表软件\08代码\Den

图 18-34 文件列表

图标为“□”的文件为没有签出的文件，而图标为“☑”的文件就是已经被签出的文件，后面还列出了签出该文件的用户名以及签往的工作目录。

(3) 撤销签出

文件被签出后，可以撤销签出操作（Undo Check Out），就好像什么都没发生一样。

- 对目录进行撤销签出操作

在 VSS 目录树状列表中单击快捷菜单中的“Undo Check Out”，会显示如图 18-35 所示的对话框。

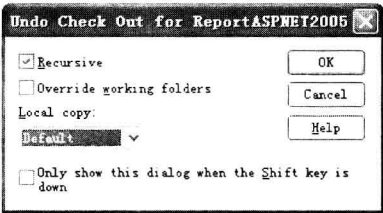


图 18-35 撤销签出目录对话框

在该对话框中，若勾选 Recursive 复选框，则对该 VSS 目录及所有子孙目录中的所有文件进行撤销签出操作。

这里设置 Local copy 的选项就是设置如何处理本地文件。其可选项为：

Default：默认操作，也就是替换。

Replace：替换。用 VSS 服务器中的文件内容替换本地文件的内容。若用户在签出期间修改了文件内容，则该操作会导致文件内容被还原到原先的内容，造成数据丢失。

Leave：保留。保留本地文件中的内容，不做内容还原操作。

Delete：删除。删除本地文件。

撤销签出操作不会将本地文件内容上传到 VSS 服务器上，而且修改 VSS 文件的状态为未签出，并设置本地文件属性为只读。

- 对文件进行撤销签出操作

在 VSS 文件列表中选择一个或多个文件，然后单击快捷菜单中的“Un Check Out”，会显示如图 18-36 所示的对话框。

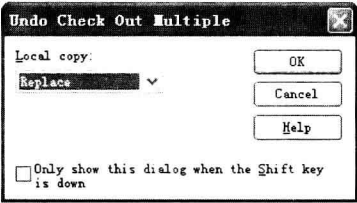


图 18-36 撤销签出文件对话框

在该对话框中，用户设置“Local copy”中的项目，然后单击“OK”按钮即可对当前选中的 VSS 文件完成撤销签出操作。

(4) 获取最新版本

当 VSS 服务器中的文本被其他用户修改后，用户可以使用获取最新版本（Get Latest

Version) 的操作来下载 VSS 服务器上的文件内容，以便更新本地文件的内容，对于已经被其他用户签出的文件也能获取最新版本。若 VSS 文件被当前用户签出了，而当前用户又修改了本地文件内容，则获取的最新版本将覆盖用户的修改，造成数据丢失。

- 对目录获取最新版本

在 VSS 目录树状列表中，单击快捷菜单中的“Get Latest Version”菜单项目，会显示如图 18-37 所示的对话框。

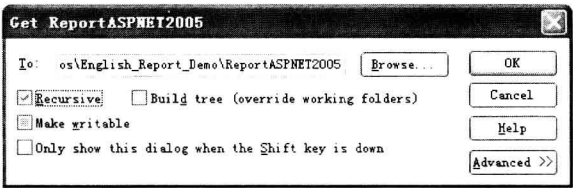


图 18-37 获取目录最新版本

在该对话框中，若勾选 Recursive 复选框，则系统会遍历所有的 VSS 子孙目录，将其所有的文件内容都下载到本地；若勾选 Build tree 复选框，则系统会根据需要创建新的本地目录，以保持 VSS 目录都有相对应的工作目录；若勾选 Make writable 复选框，则系统会设置更新过内容的本地文件为可读写状态，否则设置为只读状态。

- 对文件获取最新版本

在 VSS 文件视图列表中，单击快捷菜单中的“Get Latest Version”菜单项目，会显示如图 18-38 所示的对话框。单击“OK”按钮就可以获得指定文件的最新版本。

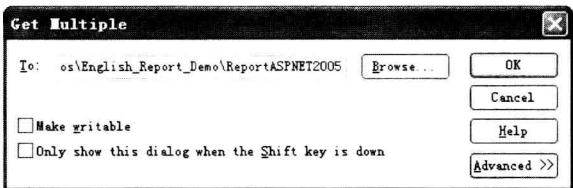


图 18-38 获取文件最新版本

(5) 签入

VSS 文件被签出后，其对应的本地文件被标记为可读写，此时若用户修改了本地文件内容，则用户可以使用签入 (Check In) 操作将本地文件内容上传到 VSS 服务器上，更新 VSS 文件的内容。这样其他用户便可以通过获取最新版本的操作来获得这个用户修改后的文件内容。

签入操作只处理被当前用户签出的文件，不处理被其他用户签出的文件。

- 对目录执行签入操作

在 VSS 目录节点的快捷菜单中单击“Check In”菜单项目，会显示如图 18-39 所示的对话框。

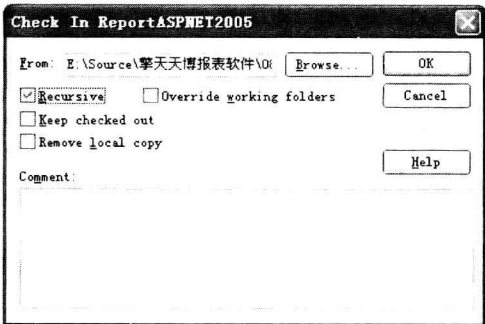


图 18-39 签入目录

若勾选 Recursive 复选框，则对所有子孙目录下的所有被当前用户签出的文件执行签入操作。单击“OK”按钮即可开始执行签入文件操作。

- 对文件执行签入操作

单击 VSS 文件项目快捷菜单中的“Check In”菜单项目，会显示如图 18-40 所示的对话框。单击“OK”按钮即可开始执行签入文件操作。

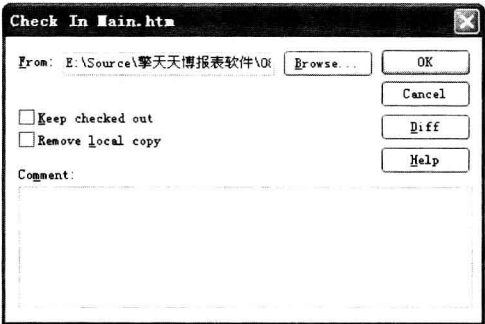




图 18-40 签入文件

7. 其他功能

(1) 打标签

打标签就是给 VSS 的文件或目录附加一个指定内容的标签，使配置管理员能快速查找到打了标签的 VSS 文件或目录，以便进行统一处理。

单击 VSS 目录节点快捷菜单中的“Label”菜单项目，或者单击工具条上的按钮“Label version”，系统会显示如图 18-41 所示的对话框。在该对话框中，用户可以在 Label 文本框中输入标签内容。

VSS 文件的快捷菜单中没有“Label”菜单项，只能通过单击工具条上的“Label version”按钮来打标签。

(2) 显示差别

VSS 软件客户端提供 VSS 文件或目标与当地文件或目录直接的差别。

- 显示目录的差别

单击 VSS 目录节点快捷菜单中的“Show Differences”菜单项目，会显示如图 18-42 所示的对话框。

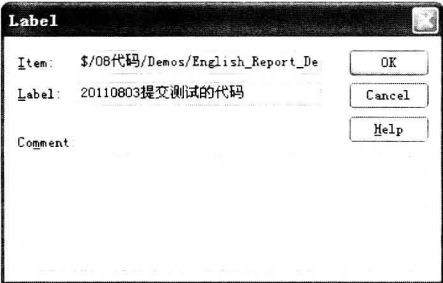


图 18-41 标签对话框

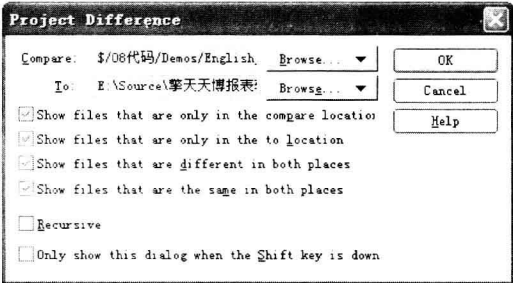


图 18-42 比较目录的差别

在该对话框中设置一些参数，例如要参与比较的 VSS 目录地址和本地目录地址，设置后单击“OK”按钮，系统会显示如图 18-43 所示的对话框：

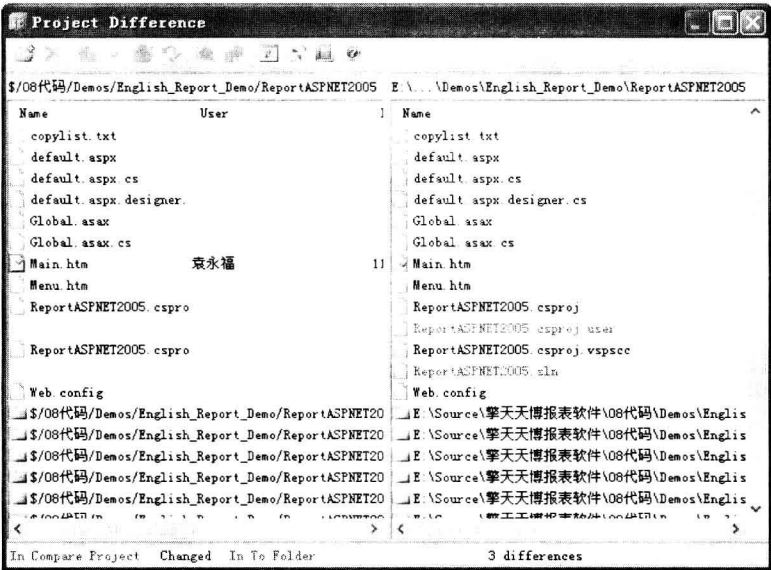


图 18-43 比较结果

在该对话框中，用户可以看到左边列出了 VSS 目录下的文件和子目录；右边列出了本地工作目录下的文件和子目录的对比情况，其中绿色显示的项目是本地存在的但没有参与 VSS 源代码管理的文件或目录，此时可以将这些项目加入到 VSS 源代码管理中。

- 显示文件的差别

单击 VSS 文件快捷菜单中的“Show Differences”菜单项目，会显示如图 18-44 所示的对话框。

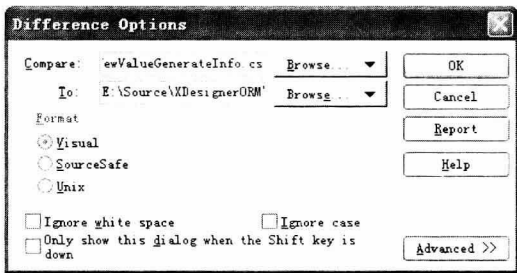


图 18-44 比较文件的差别

在该对话框中，可以对文本内容比较进行一些设置，单击“OK”按钮后，系统会显示如图 18-45 所示的对话框。

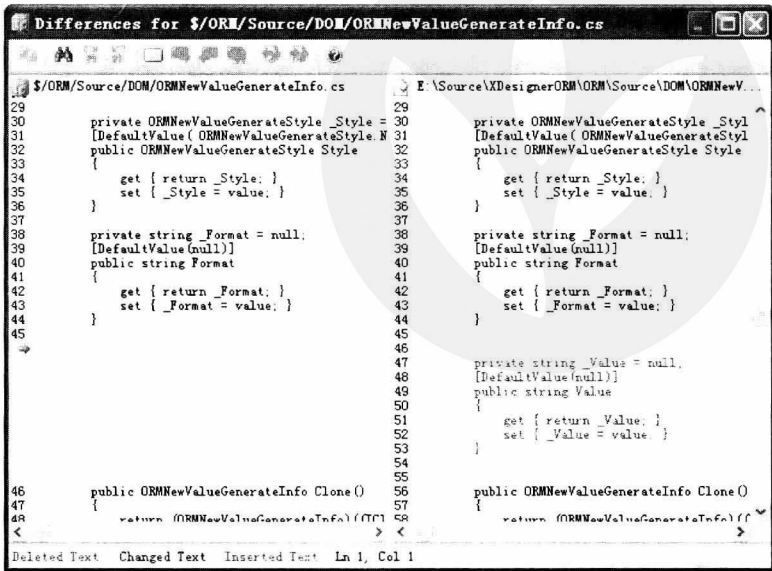


图 18-45 比较结果

在该对话框中，左边显示的是服务器中 VSS 文件的内容，右边显示的是本地文件内容。VSS 客户端软件会分析这两个文件的内容，并以一些方式突出显示两者之间的差别。

(3) 查看操作历史记录

在 VSS 客户端软件中，可以查看对 VSS 文件或目录进行操作的历史记录。

- 查看目录的操作历史记录

单击 VSS 目录快捷菜单中的“Show History”菜单项目，会显示如图 18-46 所示的对话框。

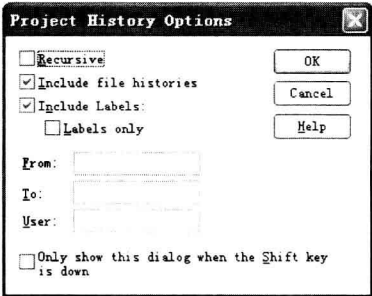


图 18-46 查看目录历史记录

在该对话框中，勾选 Recursive 复选框可查看该 VSS 目录下所有子孙目录的操作历史记录；勾选 Include file histories 复选框将显示所有文件的操作历史记录，包括签入、签出的操作记录；勾选 Include Labels 复选框将显示打标签的操作历史记录。

用户单击“OK”按钮后，系统会显示如图 18-47 所示的历史记录清单。该清单列出了该目录下所有的操作历史记录。

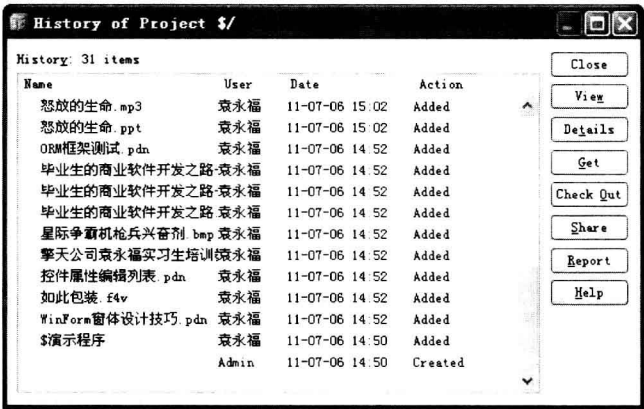


图 18-47 历史记录清单

- 查看文件的操作历史记录

单击 VSS 文件快捷菜单中的“Show History”菜单项目，会显示如图 18-48 所示的对话框。

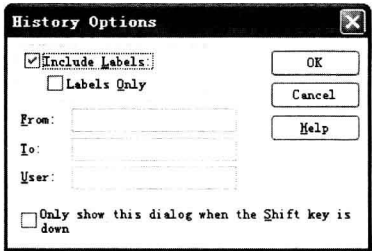


图 18-48 查看文件历史记录

单击“OK”按钮会弹出如图 18-49 所示的对话框，其中列出了当前 VSS 文件操作历史记录。

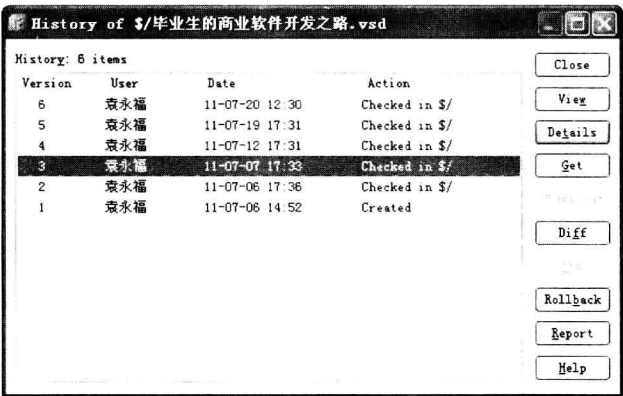


图 18-49 历史记录清单

在该对话框中，用户可以单击“Diff”按钮来显示历史上的某个版本的 VSS 文件内容与本地文件内容的比较结果；单击“Rollback”按钮可以回滚操作，将 VSS 文件恢复到此前的某个状态。

18.2.4 在 VS.NET 中使用 VSS

VSS 源代码管理软件已经和 VS.NET 紧密地集成在一起了，当 VS.NET 中打开的工程文件参与了源代码管理时，开发人员可以在 VS.NET 的用户界面中进行源代码的版本管理，而无须另外打开 VSS 客户端软件。

若系统中没有安装 VSS 源代码管理软件，则 VS.NET 中没有源代码管理功能，其用户界面也没有相关的按钮和菜单项。

1. 添加开发工程到源代码服务器中

在 VS.NET 开发环境中，在解决方案资源管理器中选某个尚未参与源代码管理的项目节点，如图 18-50 所示。



图 18-50 解决方案资源管理器

单击主菜单项目“文件→源代码管理→将选择的项目添加到源代码管理”，会弹出如图 18-51 所示的 VSS 源代码服务器登录对话框。



图 18-51 VSS 源代码服务器登录对话框

在该对话框中选择正确的 VSS 数据库，输入用户名和密码后单击“OK”按钮，会显示如图 18-52 所示的 VSS 路径选择对话框。

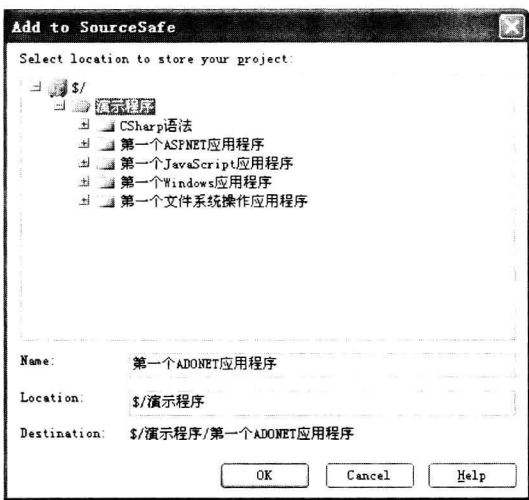


图 18-52 VSS 路径选择对话框

在该对话框中选择某个目录，然后单击“OK”按钮关闭对话框，VS.NET 就会将当前选择的工程中的所有文件保存到 VSS 目录下，并根据本地文件目录结构在 VSS 中自动生成相应的文件目录结构，同时将所有文件设置为签入状态。

在图 18-52 中，若选中的 C# 项目名为“第一个 ADONET 应用程序”，则系统会自动地在 VSS 目录“\$/演示程序”下创建“第一个 ADONET 应用程序”的子目录。

这样就完成了在 VS.NET 中将已有的项目添加到 VSS 源代码管理中。此时在 VS.NET 的解决方案资源管理器中，可以看到该项目文件，如图 18-53 所示。该 C# 工程添加到源代码管理

后，所有的文件节点前面都额外附加了一个小锁图标，这个图标表示了文件的 VSS 源代码管理状态。

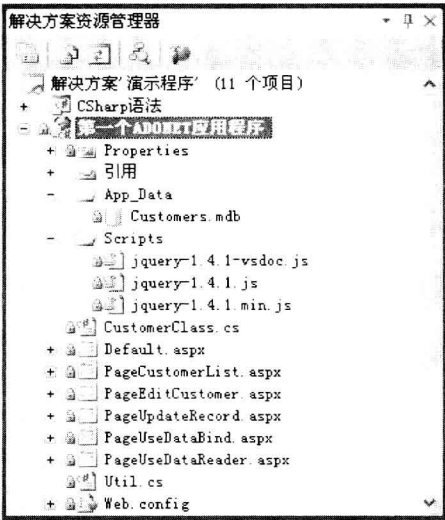


图 18-53 解决方案资源管理器

参与源代码管理的项目文件前面附加的图标其含义如表 18-2 所示。

表 18-2 图标含义

图 标	VSS 状态
	签入
	签出
	新增的文件，尚未添加到 VSS 数据库中

2. 更改源代码管理

单击 VS.NET 主菜单项目“文件→源代码管理→更改源代码管理”，程序会显示如图 18-54 所示的对话框。

在该对话框中能更改整个解决方案中各个项目的源代码管理配置。该对话框上面是工具条，中间是解决方案中各个项目的列表，在这个列表中，若项目没有参与源代码管理，则其服务器名、服务器绑定字段都是“<没有服务器>”，而且其状态为“不受控制”。

该对话框上面的工具条中，常用按钮的功能如下：

浏览：单击该按钮会弹出一个 VSS 登录对话框和 VSS 目录选择对话框，能更改列表中选择 的程序工程项目使用的 VSS 服务器名和 VSS 路径名。

取消绑定：单击该按钮会让列表中选择的项目退出源代码管理。

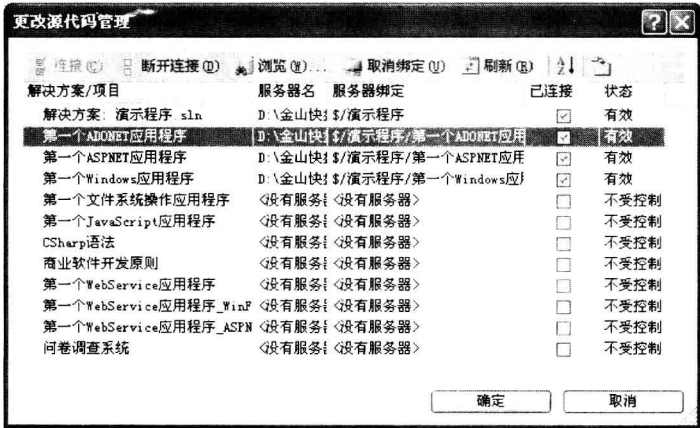


图 18-54 “更改源代码管理”对话框

3. 获取最新版本

当工程文件参与源代码管理后，在解决方案资源管理器中，用鼠标右击文件节点，会弹出如图 18-55 所示的快捷菜单。

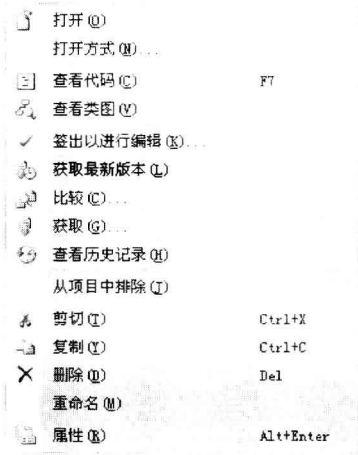


图 18-55 快捷菜单

用鼠标右击文件目录节点，会弹出如图 18-56 所示的快捷菜单。

在该快捷菜单中会出现几个与源代码管理相关的菜单项目，其中就有“获取最新版本”，单击该菜单项目会对这个文件或目录进行获取最新版本的操作。对文件目录获取最新版本的操作是指通过递归获取这个目录及其子目录下所有文件的最新版本。

4. 签出

在解决方案资源管理器中，单击某个处于签出状态的文件的快捷菜单中的“签出以进行编

辑”菜单项目，会显示如图 18-57 所示的对话框。

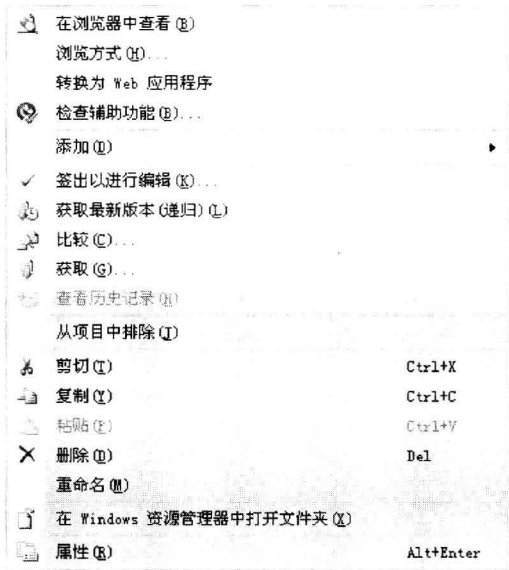


图 18-56 快捷菜单

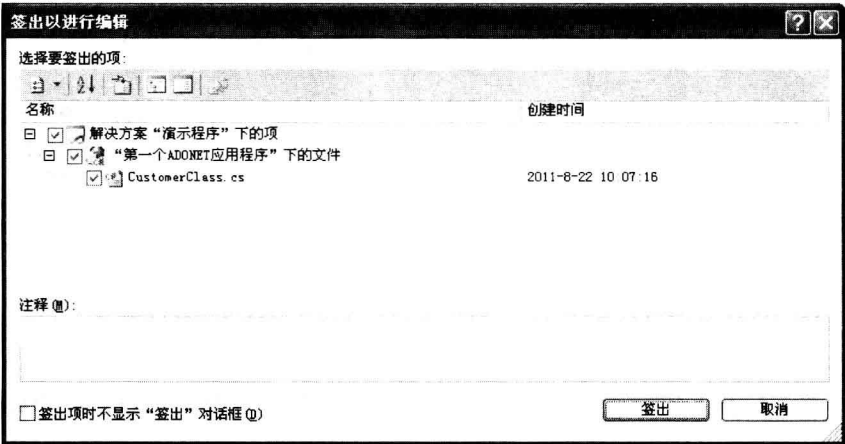


图 18-57 “签出以进行编辑”对话框

在该对话框中单击“签出”按钮，会从 VSS 服务器中签出该文件，并将本地文件从只读状态更改为可读写状态，开发者就可以修改该文件中的内容了。

对某个目录或程序工程本身执行签出操作，程序会显示如图 18-58 所示的对话框。

在该对话框中列出了该目录及其子目录下所有处于签入状态的文件，用户可以在要签出的文件前面打钩，单击“签出”按钮就能签出所有勾选的文件。

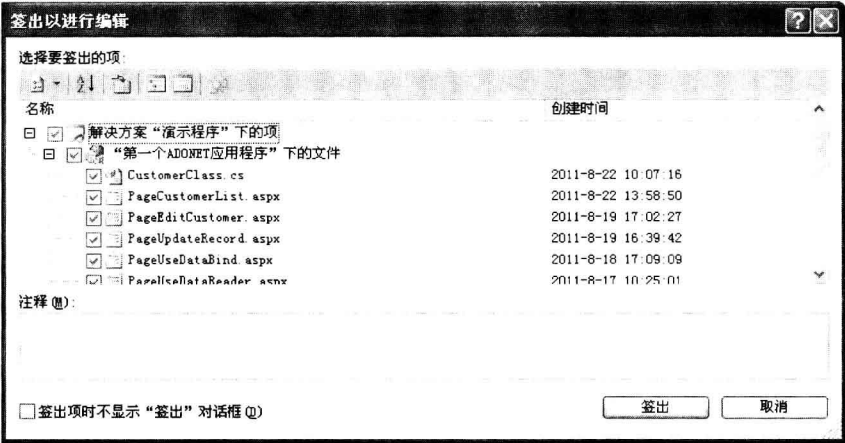


图 18-58 “签出以进行编辑”对话框

5. 签入

在解决方案资源管理器中，单击某个处于签出状态的文件的快捷菜单中的“签入”菜单项目，会显示如图 18-59 所示的对话框。

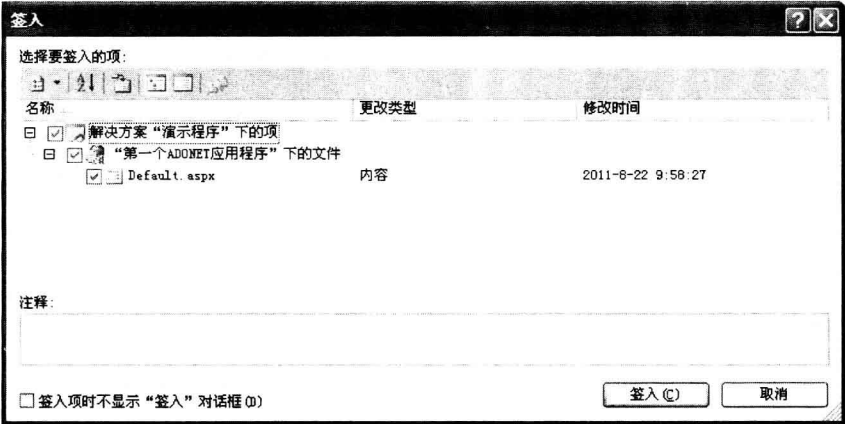


图 18-59 “签入”对话框

在该对话框中单击“签入”按钮，程序会将本地文件签入到 VSS 服务器中，上传文件内容，并设置本地文件为只读状态。

单击工程项目或文件目录的快捷菜单中的“嵌入”菜单项目，程序会显示如图 18-60 所示的对话框。

该对话框中列出了目录和各级子目录中所有处于签出状态的文件，用户勾选文件，单击“签入”按钮就能将所有选中的文件签入到 VSS 服务器中。

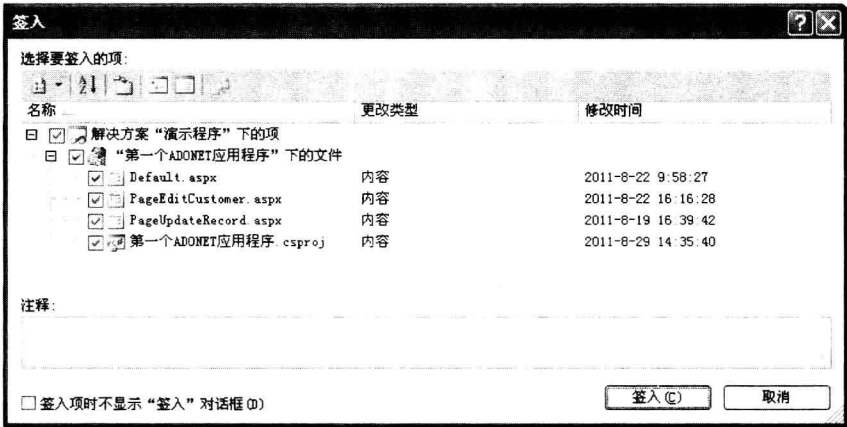


图 18-60 “签入”对话框

6. 撤销签出

在解决方案资源管理器中，单击快捷菜单中的“撤销签出”菜单项目，程序会显示如图 18-61 所示的对话框。

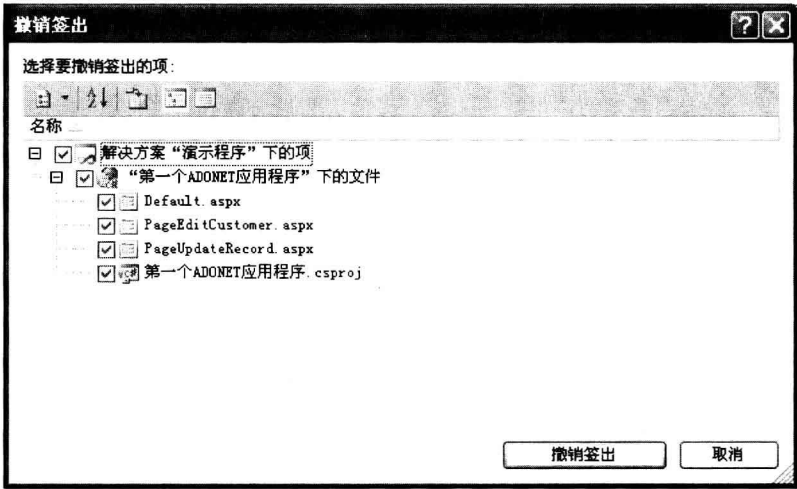


图 18-61 “撤销签出”对话框

该对话框列出了所有可以执行撤销签出的文件，用户可以通过勾选，然后单击“撤销签出”按钮来对所勾选的文件执行撤销签出操作。

源代码管理是商业软件开发管理中的基础技术手段，任何商业软件开发人员，不管使用什么软件开发技术，包括 C#、Java 等，都需要使用源代码管理工具来进行团体开发。而对于使用 VS.NET 的 C# 开发者，最方便的就是使用 VSS 源代码管理工具了。

关于企业培训

本附录介绍企业培训的意义，首先引用一个历史案例。

A.1 案例：章邯练兵

秦朝末期陈胜、吴广起义，发展迅速，各地纷纷响应。公元前 208 年，陈胜派遣的周章率兵几十万突然攻入函谷关，距离秦都咸阳几十里，秦朝束手无策。此时少府章邯请求朝廷赦免了附近 70 万骊山刑徒，发给兵器，临时组成一支军队迎击周章，将其赶出函谷关。

初次胜利后章邯并没有立即追出函谷关，而是在国难当头、皇帝催战的情况下，顶住压力，花了 2 个月的时间训练这只军队，形成一定的战斗力，然后才出函谷关到处征战，镇压了陈胜吴广等多路起义军。后来遇到项羽，虽败但未降，最后因为后方朝廷内斗而被迫向项羽投降。

可以说章邯颇具将才和魄力，在万般困难的情况下用 2 个月的时间换来了秦朝 2 年的寿命，若没有朝廷内斗，说不定还能让秦朝翻盘。如果章邯没有审时度势，不抽出时间训练军队而仓促出关，很容易被各路起义军围歼，秦朝会立即灭亡。

在此引用这个案例是想说明一个道理：训练对组织的战斗力是至关重要的，是值得花时间花精力去做的。所有不开展员工培训的理由都是借口。人家章邯都快亡国了还能抽出那么长的时间训练军队，脚踏实地地做企业的就没有理由不紧抓对员工进行培训。

A.2 企业培训的意义

有不少企业，尤其是小企业，出于成本考虑，很少关心对企业内员工的培训，特别是不愿花钱花精力进行新员工培训。对于只图赚眼前利益的企业，显然这样能短期内节省成本，但长期看成本是巨大的，得不偿失。没有足够的培训就没有“战斗力”，面对竞争对手和客户时在技术服务等方面就没有什么能拿得出手的。这样会形成“没有培训→员工技能差→企业技术弱→影响市场竞争→效益差→压缩成本→没有培训”的恶性循环，最终导致大量的技术人才流失。因此企业需要对员工进行全方位的培训。

有一个案例，某新人参与了项目开发，结果在编写删除数据的功能时，Delete SQL 语句后面

没加查询条件，导致客户数据丢失，项目组全体人员都被扣奖金。

有种观点认为公司员工不需要集中起来进行具体的技术培训，应该指导他们自己摸索学习新技术，在自发学习的过程中不断锻炼出自觉探索的能力，激起自发的对软件开发的兴趣。具体的技术培训虽然有点拔苗助长、代替学员独立思考的嫌疑，但还是很重要的，尤其是在中国国情下。

(1) 从管理上看，对新员工集中培训能提高企业的效益。

现在很多企业采用传统的传帮带的方式，由老员工带领新员工，企业对新员工是“散养”的，这会导致一些问题。首先是教出来的新员工的水平很大程度上依赖老员工的水平，如果老员工能力强、传帮带细致、认同企业文化、执行力好，则带出的新员工也比较强；而一些老员工能力差又比较油，则带出的新员工的生产力就比较低。

另外，对公司的代码规范、开发标准等，不同的老员工对其理解也不一致。比如老员工甲对公司的开发标准理解误差有 10%，则带出的新员工对其理解的误差可能就会有 20%；而老员工乙对标准理解的误差有 20%，则带出的新员工的误差可能就会有 30%。

以上原因导致普通老员工带出普通新员工，而精英老员工带出精英新员工，这样就造成传帮带出来的新员工水平参差不齐，不利于管理。

而实行集中培训，让公司最好的精英老员工对新员工进行培训，这样公司就能获得更多的精英新员工，新员工对公司文化、开发标准等都能有比较好的理解，水平差别不太大，便于管理。

(2) 从成本上看，对新员工集中培训能降低公司的成本。

比如某企业有 50 个老员工、50 个新员工，每个老员工带一个新员工，每个老员工都需要耗费 10 点精力来传帮带新员工，这样企业为培养新员工整体上消耗了 $50 \times 10 = 500$ 点精力。

若让一个老员工耗费 50 点精力对新员工进行集中培训，这样每个老员工只要消耗 5 点精力来传帮带新员工，那么企业为培养新员工整体上消耗了 $50 + 50 \times 5 = 300$ 点精力。

通过比较，新员工集中培训能为公司节省 $500 - 300 = 200$ 点精力，而员工的每点精力都是公司花钱购买的，因此通过集中培训能降低公司的人力资源成本。

(3) 从时间上看，对新员工的集中培训能缩短培训的时间。

从企业的角度看，毕业生尚未形成实际的生产力，但还要照常发工资，这是不小的成本，因此需要尽快地让毕业生具备实际的生产力，而具体的技术培训就能帮助缩短这个过程。对于培养毕业生员工的探索、创新能力，大部分企业认为这不是当务之急。的确，做事要有先后顺序。

从毕业生的角度看，竞争是激烈的，只有越早形成实际生产力，才能在人才市场上处于优势，为企业所青睐。为此急需提升自己的实际生产力，而培训对此就能提供一些帮助。经过系统的培训，能从一开始就养成良好的软件开发习惯，这样比以后再改掉坏习惯要好得多。

A.3 学校教育和企业培训

一个优秀的企业必然需要一批充满探索、创新能力的优秀员工。这种员工在我国不太多。因

为学校的教育尤其是中小学教育有一部分仍是以考试分数为纲的填鸭式教育。在这种教育制度下，学生的创新能力普遍不高。

有人统计过，在 1977 年至 2008 年的 1000 余位高考“状元”中，几乎没出现做学问、经商、从政等方面的顶尖人才，他们的职业成就远低于社会预期。而那些成绩平平，能独立思考的学生让人刮目相看的太多。

例如，笔者在参加一些毕业生员工的培训实践中，发现双方进行互动有些困难，即使采取某些手段也收效不大。事后询问时有的学生说可能受正规教育压抑惯了，没有互动的习惯。

但是企业中的培训不是以考试成绩为纲的，已经脱离正规教育制度，因此可以完全抛弃填鸭式的教育方式，而采用科学的培训方式。在传师授道的同时，启发新员工对技术、对工作的激情，形成创新型生产力，这对企业、对员工都有莫大的好处。

下面介绍某保险公司如何对新员工进行集中封闭培训。

卖保险是一个很艰难的职业，经常遭到客户的严词拒绝，要干下去就得有持久的激情，否则无法做下去。保险公司拿出两种方式让员工保持生产力，一个是物质奖励，另一个是精神渲染。

保险公司对员工的物质奖励名目繁多，有现金，有公费旅游，等等，只要干得好，就能有一笔又一笔的奖励，这种物质奖励制度放在软件公司里是不可想象的。

保险公司对员工的精神渲染也是很厉害的。每天必开早会，经理气宇轩昂地宣传公司理念，近期表现好的员工上台介绍自己的销售经验，公司走道中贴满了有关员工业绩排名的宣传单。所有的一切都在刺激员工，使员工保持一种充满激情的状态，拼命地工作，拼命地卖保险。相信有不少人对保险销售员反反复复的推销活动印象深刻。如果企业的员工都具有保险销售员一样的激情，那么企业发展不成问题。

对于企业培训，除了具体的职业技能传授外，还需要进行精神激励，提高员工对工作的激情；特别是毕业生，毕竟是年轻人，血气方刚，调动他们的激情还是比较容易的。若员工对工作充满激情，那么企业中什么事都好办，创新、发展都不是问题，保守主义、官僚主义也会消失。因此企业中的培训特别需要调动学员的激情。

关于盗版

讲盗版的问题是想让初进软件行业的毕业生对盗版有个明确的概念，了解中国软件行业深受盗版的毒害，必须在盗版的阴影下摸索出生存和发展的道路。

盗版是有害的，是跨国软件巨头对中国软件产品的倾销，这是一场寂静的战争，会毁掉我们中国的软件行业。

软件产品大体分为系统软件和行业应用软件两类。系统软件包括操作系统、数据库、中间件等；行业应用软件就是针对特定行业应用而开发的软件，比如电信的缴费系统、企业中各种办公自动化系统等。行业应用软件是建立在系统软件之上的，系统软件是整个软件行业的基础，同时也是利润最大的部分。

某个版本的 SQL Server 数据库系统价格包括：一个 CPU 是 25 万人民币，某个版本的 Oracle 数据库系统是 35 万；而一个小型系统的应用软件价格可能只有 20 万。某银行的信息化系统，总合同金额达一百多万，但其中的应用软件的价格只有 3 万，其他的都是服务器和数据库系统的价格。虽然中国的软件市场销售额比较大，但跨国软件企业在其中赚取了大部分利润，其他应用软件企业赚的都是微薄的辛苦钱。

跨国软件公司为了追求其在中国市场中的利益最大化，就不可能让中国软件行业独立，只能依赖他们。使中国完全不能开发软件那是不可能的，也不符合他们的利益，于是允许中国发展行业应用软件和软件外包业，严厉打击中国的系统软件，这是需要我们认真对待的问题。

B.1 案例：《多收了三五斗》

叶圣陶先生曾经写过一篇著名的《多收了三五斗》的文章，深刻描述了旧中国农民破产的情景。以下摘取了该文章的部分内容：

万盛米行的河埠头，横七竖八停泊着乡村里出来的敞口船。船里装载的是新米，把船身压得很低。齐般舷的菜叶和垃圾给白腻的泡沫包围着，一漾一漾地，填没了这船和那船之间的空隙。

河埠上去是仅容两三个人并排走的街道。万盛米行就在街道的那一边。早晨的太阳光从破了明的瓦天棚斜射下来，光柱子落在柜台外面晃动着的几顶旧毡帽上。

那些戴旧毡帽的大清早摇船出来，到了埠头，气也不透一口，便来到柜台前面占卜他们的命

运。

“糙米五块，谷三块，”米行里的先生有气没力地回答他们。

“什么！”旧毡帽朋友几乎不相信自己的耳朵。美满的希望突然一沉，一会儿大家都呆了。

“在六月里，你们不是卖十三块么？”

“十五块也卖过，不要说十三块。”

“哪里有跌得这样厉害的！”

“现在是什么时候，你们不知道么？各处的米像潮水一般涌来，过几天还要跌呢！”

刚才出力摇船犹如赛龙船似的一股劲儿，现在在每个人的身体里松懈下来了。今年天照应，雨水调匀，小虫子也不来作梗，一亩田多收这么三五斗，谁都以为该得透一透气了。哪里知道临到最后的占卜，却得到比往年更坏的课兆！

“还是不要粜的好，我们摇回去放在家里吧！”从简单的心里喷出了这样的愤激的话。

“嗤，”先生冷笑着，“你们不粜，人家就饿死了么？各处地方多的是洋米，洋面，头几批还没吃完，外洋大轮船又有几批运来了。”

洋米，洋面，外洋大轮船，那是遥远的事情，仿佛可以不管。而不粜那已经送到河埠头来的米，却只能作为一句愤激的话说说罢了。怎么能够不粜呢？田主方面的租是要缴的，为了雇帮工，买肥料，吃饱肚皮，借下的债是要还的。

.....

第二天又有一批敞口船来到这里停泊。镇上便表演着同样的故事。这种故事也正在各处市镇上表演着，真是平常而又平常的。

这篇《多收了三五斗》貌似小家子气地详细描述了微观细节的事情，但却深刻地展示了一个宏观现象，那就是西方列强对落后国家的商品倾销。

B.2 案例：《管子》

再举一个案例，战国时的书籍《管子》中曾有如下记载：

齐桓公问管仲曰：“鲁梁之于齐也，千谷也，蜂螫也，齿有唇也。今吾欲下鲁梁，何行而可？”管子对曰：“鲁梁之民俗为绀（念替），公服绀，令左右服之，民从而服之。公因令齐勿敢为，必仰于鲁梁，则是鲁梁释其农事而作绀矣。”桓公曰：“诺。”即为服于泰山之阳，十日而服之。管子告鲁梁之贾（念古）人曰：“子为我致绀千匹，赐子金三百斤，什至而金三千斤，则是鲁梁不赋于民而财足用也。”鲁梁之君闻之，则教其民为绀，十三月而管子令人之鲁梁，鲁梁郭中之民，道路扬尘，十步不相见，继（念泄）繻（念桥）而踵相随，车轂（念轱）𨾏（念邹），骑连伍而行。

管子曰：“鲁梁可下矣。”公曰：“奈何？”管子对曰：“公宜服帛，率民去绀闭关，毋与鲁梁

通使。”公曰：“诺”。后十月，管子令人之鲁梁，鲁梁之民，饿馁相及，应声之征，无以给上。鲁梁之君，即令其民去缁修农谷，不可以三月而得。鲁梁之人余（念敌）十百，齐巢（念跳）十钱。二十四月，鲁梁之民归齐者十分之六，三年，鲁梁之君请服。

这段话的意思是，齐国的国君齐桓公想吞并旁边的鲁国和梁国，于是管仲献计说：鲁国和梁国都有纺织的传统，现在请大王带头让全体齐国人穿鲁国和梁国制造的衣服，而且还高价进口，使鲁国和梁国放弃农业，一心搞纺织赚钱。齐桓公采纳了这个建议。

管仲又跑去告诉鲁国和梁国的人，说齐国重金大量进口衣服，鲁国和梁国可以因此发大财，于是鲁国和梁国的国君大力发展纺织业，荒废了农业，而从齐国进口粮食。

过了一段时间，管仲又建议齐桓公禁止穿鲁国和梁国制造的衣服，断绝与鲁国和梁国的任何交往。不久鲁国和梁国没有粮食了，虽然鲁国和梁国的国君命令开始发展农业，但粮食不是短期内所能生产的，于是鲁国和梁国闹饥荒，大批民众跑到齐国，不久鲁国和梁国均臣服于齐国。

B.3 案例：金山软件公司

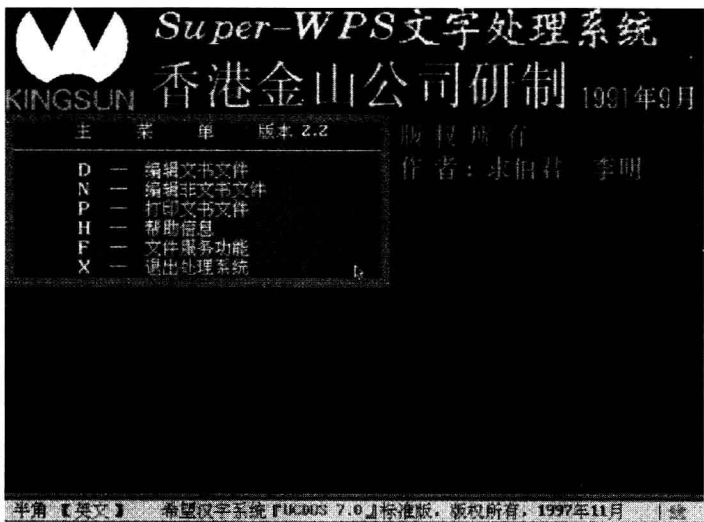
下面根据上面引用的案例来分析中国的软件行业。其典型案例就是金山软件公司。

提到金山软件公司，现在很多毕业生想到的是网络游戏。是的，现在的金山公司确实以网络游戏为主要业务。图 B.1 是 2011 年 6 月 1 日金山公司官方网站的截屏。



B.1 金山官方网站

其实金山公司当年是做通用软件产品的。金山软件股份有限公司是 1988 年成立的，当年它的拳头产品是 WPS 文字处理系统。图 B.2 是 WPS 文字处理系统的截屏。



B.2 WPS 文字处理系统

在上世纪末，WPS 在中国非常风靡，所有的打字社都用 WPS 来编辑文档和打印，中国计算机等级考试中考的也是 WPS，甚至连微软的 Word 文字处理软件也落在它的后面。

金山软件公司是中国历史最悠久的纯软件企业，也是国内少见的几十年如一日地执着追求软件技术的软件企业。这使得金山公司积累了非常强大的技术力量，有能力开发出中国的通用软件，其董事长求伯君也成为最出名的中国软件人士，是当时软件从业人员的标杆。

不过后来在盗版、微软的操作系统的平台优势和软件倾销的多重打击下，且由于金山公司自身的经营出现了一些问题，使得微软的 Word 文字处理软件很快占据了在中国文字处理软件市场，WPS 软件在市场上受到严重挤压一直到现在。

微软公司已经在西方市场上盈利了，拿得出几百亿美元来搞研发、搞市场，它已经不在乎中国的盗版了，反而利用盗版将中国的竞争对手扼杀在摇篮中。中国 MS Word 盗版光盘只买 4 元人民币，这是金山公司所无法接受的，于是 WPS 软件“奄奄一息”，已经不是靠企业所能重新振兴的了。

在官方网站上，金山公司一直标榜“做世界一流的软件企业”，但它已经不是一家软件企业了，口号应该改叫“争做世界一流的网游企业”。

金山公司从中国第一的软件企业沦落到网游运营企业，这不只是一家软件企业迷失了自我，而是中国通用软件企业的集体沦落的一个典型，这是中国软件行业环境的必然结果。而中国的通用文字处理软件已经被微软绝对垄断了。没有微软，我们不能写电子文档，做电子表格，开发应用系统；微软、IBM、ORACLE 等跨国公司已经牢牢地掌握了中国软件行业的命脉。

B.4 倾销的定义

根据百度百科的定义，倾销，是指一国的生产商或出口商以低于其国内市场价格或低于成本价格将其商品抛售到另一国市场，从而使得另一国国内有竞争能力的产业受到损害的行为。其构成要件如下：

(1) 产品以低于正常价值或公平价值的价格销售。

(2) 这种低价销售的行为给进口国产业造成损害，包括实质性损害、实质性威胁和实质性阻碍。

(3) 损害是由低价销售造成的，两者之间存在因果关系。

倾销通常具有以下特征：

(1) 倾销是一种人为的低价销售措施。它由出口商根据不同的市场，以低于有关商品在出口国的市场价格对同一商品进行差价销售。

(2) 倾销的动机和目的是多种多样的，有的是为了销售过剩产品，有的是为了争夺国外市场，扩大出口，但只要对进口国某一工业的建立和发展造成实质性损害或实质性威胁或实质性阻碍，就会招致反倾销措施的惩罚。

(3) 倾销是一种不公平竞争行为。在政府奖励出口的政策下，生产者为了获得政府出口补贴，往往以低廉价格销售产品；同时，生产者将产品以倾销的价格在国外市场销售，从而获得在另一国市场的竞争优势并进而消灭竞争对手，再提高价格以获取垄断高额利润。

(4) 倾销的结果往往对进口方的经济或生产者的利益造成损害，特别是掠夺性倾销扰乱了进口方的市场经济秩序，给进口方经济带来毁灭性打击。

B.5 盗版即倾销

下面按照定义逐条来分析盗版到底是不是软件产品倾销。

B.5.1 盗版的表面现象

首先说明表面现象，2010 年 6 月 16 号，在卓越亚马逊网站上，Windows 7 家庭高级版的销售价格是 679 元；而同期在南京珠江路上盗版 Windows 7 光盘销售价格是 4 元，这两张 Windows 7 光盘都可以正常安装，使用上没有差别。

Windows 7 是美国微软公司生产的软件商品，并出口到中国内地市场，因此存在跨国商品流通。

在中国内地市场，679 元是 Windows 7 操作系统的正常价格，而 4 元是远低于正常价格的，

因此若用户以 4 元购买了 Windows 7，则从价格上看有可能是倾销。

假设 A 企业用 679 元从正规经销商处购买了 Windows 7 正版光盘，那么经销商必然将这 679 元扣除所获得的提成后再转交给美国微软公司，这是一条合法的正常的软件销售途径。这是美国微软公司用符合软件价值的价格在进行产品销售。

假设 B 企业使用 4 元从地摊上购买了 Windows 7 盗版光盘，盗版商将全额占有这 4 元而不会为美国微软公司支付任何费用。这是一条非法的地下软件销售途径。

B.5.2 盗版的價格特性

从账面上看，美国微软公司不会从盗版软件的销售活动中获得任何现金利益，但无论是正版的还是盗版的 Windows 7，其版权都属于美国微软公司，都是其生产的。

软件产品只是软件知识产权的一种表现而已，消费者购买软件不是简单的购买一张光盘，而是从软件公司那里获得使用软件知识产权的使用权，就像人们握着人民币是拥有人民币的使用权，但人民币是归中国人民银行拥有的。在本案例中，消费者无论是购买正版软件还是盗版软件，都是从美国微软公司那里获得微软拥有的 Windows 7 知识产权的使用权。

消费者可以从正版商或盗版商手中购买 Windows 7 软件的使用权，因此正版商和盗版商都是美国微软公司的软件产品推广渠道，只是盗版商没有向微软公司提供分成。也可以说，美国微软公司使用零元的价格批发给盗版商，通过盗版商这条渠道来销售 Windows 7 软件，这是一种非常极端的做法。

零元的批发价是远远低于 679 元的市场合理价格的，这就形成商品倾销的价格特性。而美国微软公司确实有提供 679 元的市场合理价格的软件销售途径，这是一种明修陈仓，暗修栈道的做法。美国微软公司明着提供正规软件销售渠道，而暗地里又纵容盗版的存在，从而成功地掩盖了其软件倾销的价格特性。

B.5.3 盗版的利益分析

在 Windows 7 盗版的过程中，涉及了美国微软公司、盗版商、用户、国内软件企业的利益，下面进行各方的利益分析。

从美国微软公司的角度看，软件产品复制简单快速，美国微软公司研发出软件产品后，可以根据市场情况按需复制，不存在软件产品生产过剩的问题。美国微软公司不能从盗版中获得任何现金利益，因此其唯一的目的是扰乱市场、打击竞争对手、占据中国软件市场。

在 Windows 7 盗版的例子中，由于 679 元和 4 元这两个价格相差巨大，而中国没有切实保护知识产权，有法不依，执法不严，而且社会主流道德并不谴责使用盗版软件，此时消费者自然会花低廉的 4 元购买盗版 Windows 7，于是产生了一定的软件盗版市场。结果使微软的产品占据中国操作系统市场份额的 95% 以上，形成垄断，微软的目的达到了。

从盗版商的角度看，由于现在计算机光盘刻录技术很先进，盗版商在进货时都是按重量来计

算的，一般是 2 元至 5 元钱 1 斤光盘，然后再以 4 元一张光盘出售，其毛利润是百分之几百的，都快超过毒品了；而且由于是非法的地下商业行为，不缴税，因此实际利润非常高。

国内执法机关对盗版商的打击力度不够，盗版商的行为虽然是非法的，但在现实中其风险不大而利润较大。在利润和风险不匹配的情况下，盗版商会大肆盗版出售 Windows 7 软件来获取利润。

从消费者的角度看，A 企业用 679 元购买了正版软件，而 B 企业用 4 元购买了盗版软件。虽然 B 企业使用了非法的盗版软件产品，但在现实中被政府执法机关查办的风险很低。也就是说，B 企业冒着很小的风险节省了 675 元的成本，这促进了企业的发展；而 A 企业遵守法律，购买正版软件，加大了企业成本。很显然 B 企业和 A 企业竞争时占有软件采购成本优势，从而使 B 企业能比 A 企业更好地实现自身利益的最大化。于是遵守法律的企业花钱，而违反法律的企业获利，这形成一种逆向激励反馈循环，使得软件盗版需求日益增强。这样就形成了中国国内的供需两旺的软件盗版市场。

从中国软件厂商的角度看，盗版 Windows 7 销售价格只有 4 元人民币，而研发 Windows 7 类型的软件费用可达 100 亿美元甚至更多。中国尚无软件企业自主研发操作系统，即使研发出来投放市场，也会因 4 元的盗版 Windows 操作系统的存在而难于销售，因此中国本土操作系统软件企业很少。盗版对中国的操作系统软件企业会造成毁灭性打击。

盗版使得微软 Windows 占据了 95% 的操作系统市场份额，尽管这其中很大一部分是盗版的，但美国微软公司牢牢地确定了其在中国软件市场上的绝对垄断地位。美国微软公司今后可以利用这种垄断地位从容实现其利益最大化，打击压制竞争对手。

从上述分析可知，盗版是美国微软公司低价销售软件的手段，能帮助美国微软公司占据中国操作系统软件市场的垄断地位，对中国本土的操作系统软件企业造成毁灭性打击。因此盗版就是不折不扣的软件产品倾销。